DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
II YEAR CSE
III SEMESTER

U23CST31-COMPUTERARCHITECTURE&ORGANIZATION

UNIT I OVERVIEWAND INSTRUCTIONS

EIGHT IDEAS

1. Design for Moore'sLaw

The one constant for computer designers is rapid change, which is driven largely by Moore's Law.

- It states that integrated circuit resources double every 18–24months.
- Moore's Law resulted from a 1965 prediction of such growth in ICcapacity.
- Moore's Law made by Gordon Moore, one of the founders ofIntel.
- As computer designs can take years, the resources available per chip can easily double or quadruple between the start and finish of theproject.
- Computer architects must anticipate this rapidchange.
- Iconused:"upandtotheright"Moore'sLawgraphrepresentsdesigningforrapidchange.

2. Use Abstraction to SimplifyDesign

Both computer architects and programmers had to invent techniques to make themselves more productive.

- A major productivity technique for hardware and soft ware is to use abstractions to represent the design at different levels ofrepresentation;
- lower-level details are hidden to offer a simpler model at higherlevels.
- Icon used: abstract paintingicon.

3. Make the common casefast

- Making the common case fast will tend to enhance performance better than optimizing the rarecase.

- The common case is often simpler than the rare case and it is often easier toenhance
- Common case fast is only possible with careful experimentation andmeasurement.

- Icon used: sports car ss the icon for making the common case fast(as the most common trip has one or two passengers, and it's surely easier to make a fast sports car than a fast minivan.)

4. Performance viaparallelism

computer architects have offered designs that get more performance by performing operations in parallel. Icon Used: multiple jet engines of a plane is the icon for parallel performance.

5. Performance viapipelining

Pipelining- Pipelining is an implementation technique in which multiple instructions are overlapped in execution. Pipelining improves performance by increasing instruction throughput.

- For example, before fire engines, a human chain can carry a water source to fire much more quickly than individuals with buckets running back andforth.

- IconUsed:pipelineiconisused.Itisasequenceofpipes,witheachsectionrepresenting one stage of thepipeline.

6. Performance viaprediction

- Followingthesayingthatitcanbebettertoaskforforgivenessthantoaskforpermission, the next great idea isprediction.

- Insomecasesitcanbefasteronaveragetoguessandstartworkingratherthanwaituntil you know forsure.

- This mechanism to recover from a misprediction is not too expensive and theprediction is relativelyaccurate.

- Icon Used:fortune-teller's crystal ball ,

7. Hierarchy of memories

- Programmers want memory to be fast, large, and cheap memory speed often shapes performance, capacity limits the size of problems that can be solved, the cost of memory today is often the majority of computercost.

- Architects have found that they can address these conflicting demands with a hierarchy of memories the fastest, smallest, and most expensive memory per bit is at the top of the hierarchy the slowest, largest, and cheapest per bit is at thebottom.

- Caches give the programmer the illusion that main memory is nearly as fast as the top of the hierarchy and nearly as big and cheap as the bottom of thehierarchy.

- Icon Used: a layered triangle icon represents the memoryhierarchy.

- The shape indicates speed, cost, and size: the closer to the top, the faster and more expensive per bit the memory; the wider the base of the layer, the bigger thememory.

8. Dependability viaredundancy

Computers not only need to be fast; they need to be dependable.

Since any physical device can fail, systems can made dependable by including redundant components.

- These components can take over when a failure occurs and to help detectfailures.

- Icon Used:the tractor-trailer , since the dual tires on each side of its rear axels allow the truck to continue driving even when one tirefails.

COMPONENTS OF A COMPUTER SYSTEM

The five classic components of a computer are input, output, memory, datapath, and control. Datapath and control are combined and called as the processor. The figure 1.1 shows the standard organization of a computer. This organization is independent of hardware technology.

The organization of a computer, showing the five classic components.

1. Input writes data to memory
2. Output reads data frommemory
3. Memory stores data andprograms

4. Control sends the signals that determine the operations of the datapath, memory, input, and output.
5. Datapath performs arithmetic and logicoperations

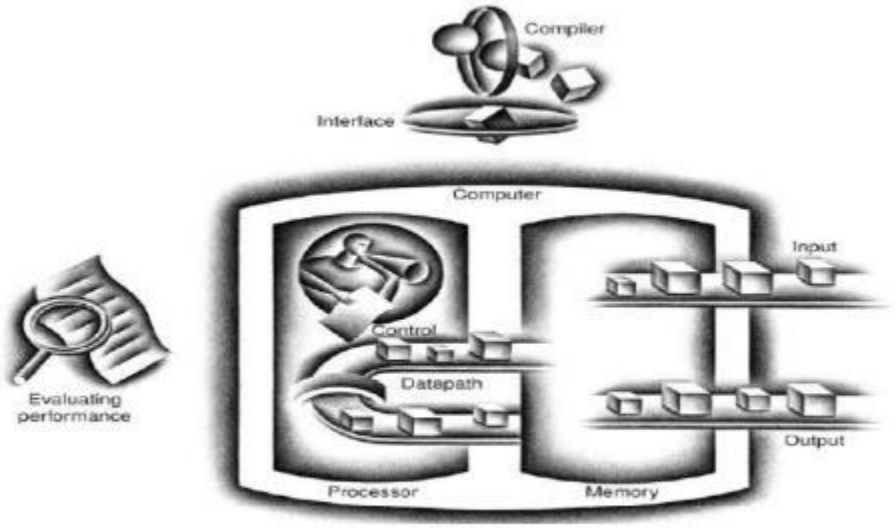


Fig.1.1 The organization of a computer

Fig.1.1 The organization of a computer

Two key components of computers:

1. **Input devices**, such as the keyboard andmouse,
2. **Output devices**, such as thescreen.

- Input device - A mechanism through which the computer is fed information, such asthe keyboard ormouse.

- Output device - A mechanism that conveys the result of a computation to a user or anothercomputer.

As the names suggest, input feeds the computer, and output is the result of computation sent to the user. Some devices, such as networks and disks, provide both input and output to the computer.

Input Device

The most commonly used input device is a keyboard. Whenever the user presses a keythe control signal be sent to the processor. The processor respond for that by displaying the corresponding character on thedisplay.

The next commonly used input device is a mouse.The original mouse was electromechanicalandusedalargeball.Whenitisrolledacrossasurface,itwouldcauseanxand y counter to be incremented. The amount of increase in each counter told how far the mouse had beenmoved.

Theelectromechanicalmousehasbeenreplacedbytheopticalmouse.Theopticalmouse is actually a miniature optical processor including an LED to provide lighting, a tiny black-and-white camera, and a simple optical processor. The LED illuminates the surface underneath the mouse;thecameratakes1500samplepicturesasecondundertheillumination.Successivepictures aresenttoasimpleopticalprocessorthatcomparestheimagesanddetermineswhetherthemouse has moved and howfar.

Output Device

The most fascinating I/O device is probably the graphics display. Most personal mobile devices use liquid crystal displays (LCDs) to get a thin, low-power display. The LCD is not the source of light; instead, it controls the transmission of light. A typical LCD includes rod-shaped molecules in a liquid that form a twisting helix that bends light entering the display, from either a light source behind the display or less often from reflected light.

Today, most LCD displays use an active matrix that has a tiny transistor switch at each pixel to precisely control current and make sharper images. As in a CRT, a red-green-blue mask associated with each pixel determines the intensity of the three color components in the final image; in a color active matrix LCD, there are three transistor switches at eachpixel.

Thecomputerhardwaresupportsarefreshbuffer,orframebuffer,tostorethebitmap.The imagetoberepresentedon-screenisstoredintheframebuffer,andthebitpatternperpixelisread out to the graphics display at the refreshrate.

Pixel - The smallest individual picture element. Screen are composed of hundreds of thousands to millions of pixels, organized in a matrix.

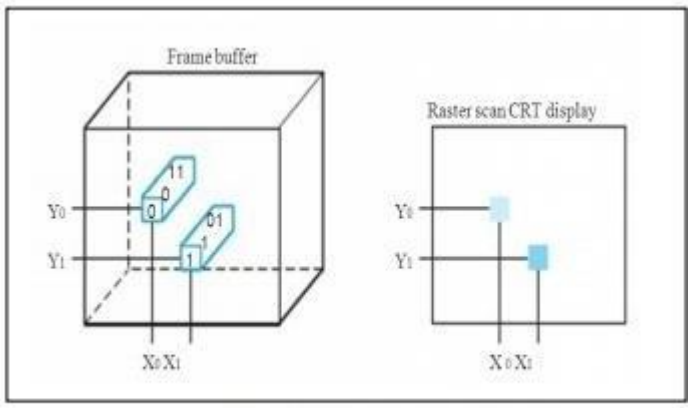


Fig.1.2 Frame Buffer

Fig.1.2 Frame Buffer

Each coordinate in the frame buffer on the left determines the shade of the corresponding coordinate for the raster scan CRT display on the right. Pixel (X0, Y0) contains the bit pattern 0011, which is a lighter shade of gray on the screen than the bit pattern 1101 in pixel (X1, Y1).

Processor

Central processor unit (CPU) - Also called processor. The active part of the computer, which contains the datapath and control and which adds numbers, tests numbers, signals I/O devices to activate, and so on. The processor gets instructions and data from memory.

- Datapath - The component of the processor that performs arithmeticoperations.
- Control-Thecomponentoftheprocessorthatcommandsthedatapath,memory,andI/O devices according to the instructions of theprogram.

Memory

- Memory The storage area in which programs are kept when they are running and that contains the data needed by the runningprograms.
- Primary memory - Also called main memory. It is a Volatile memory used to hold programs while they are running; typically consists of DRAM in today'scomputers.

- Dynamic random access memory (DRAM) Memory built as an integrated circuit, it provides random access to any locationSecondary memory Non- volatile memory used to store programs and data between runs; typically consists of magnetic disks in today's computers.

- Magneticdisk(alsocalledharddisk)Aformofnonvolatilesecondarymemorycomposed of rotating platters coated with a magnetic recordingmaterial.

- Cache memory A small, fast memory that acts as a buffer for a slower, larger memory. Cache is built using a different memory technology, static random access memory (SRAM). SRAM is faster but less dense, and hence more expensive, thanDRAM.

- Volatile memory Storage, such as DRAM, that only retains data only if it is receiving power.

- NonvolatilememoryAformofmemorythatretainsdataevenintheabsenceofapower source and that is used to store programs between runs. Magnetic disk is nonvolatile and DRAM isnot.

Removable storage technologies:

- Optical disks, including both compact disks (CDs) and digital video disks (DVDs), constitute the most common form of removablestorage.

- Magnetic tape provides only slow serial access and has been used to back up disks, in a role now often replaced by duplicate harddrives.

- FLASH-based removable memory cards typically attach by a USB (Universal Serial Bus) connection and are often used to transfer files.

- Floppy drives and Zip drives are a version of magnetic disk technology with removable flexible disks. Floppy disks were the original primary storage for personal computers, but have now largelyvanished.

TECHNOLOGY

Processors and memory have improved at an incredible rate, because computer designers have long embraced the latest in electronic technology to try to win the race to design a better computer. A transistor is simply an on/off switch controlled by electricity. Th e integrated circuit (IC)combineddozenstohundredsoftransistorsintoasinglechip.WhenGordonMoorepredicted

thecontinuousdoublingofresources,hewaspredictingthegrowthrateofthenumberoftransistorsperchip.

Todescribethetremendousincreaseinthenumberoftransistorsfromhundredstomillions, the adjective very large scale is added to the term, creating the abbreviation VLSI, for very large-scale integrated circuit. Th is rate of increasing integration has been remarkably stable. The manufacture of a chip begins with silicon, a substance found in sand. Because silicon does not conductelectricitywell,itiscalledasemiconductor.Withaspecialchemicalprocess,itispossible to add materials to silicon that allow tiny areas to transform into one of threedevices:

- Excellent conductors of electricity (using either microscopic copperor
- Excellent insulators from electricity (like plastic sheathing orglass)

- Areas that can conduct or insulate under special conditions (as aswitch)

Transistorsfallinthelastcategory.AVLSIcircuit,then,isjustbillionsofcombinationsof conductors, insulators, and switches manufactured in a single smallpackage.

PERFORMANCE

CPU performance equation.

The Classic CPU Performance Equation in terms of instruction count (the number of instructions executed by the program), CPI, and clock cycle time:

CPU time=Instruction count * CPI * Clock cycle time or

since the clock rate is the inverse of clock cycle time:

CPU time = Instruction count *CPI / Clock rate

T = N X S / R

Relative performance:

Performance A / Performance B = Execution time B / Execution time A = n

CPU execution time for a program

CPU execution time for a program =

CPU clock cycles for a program * clock cycle time or

since the clock rate is the inverse of clock cycle time:

CPU execution time for a program=
CPU clock cycles for a program / Clock rate

CPU clock cycles required for a program

CPU clock cycles = Instructions for a program * Average clock cycles per instruction

Basic components of performance

The basic components of performance and how each is measured are:

Components of Performance
CPU execution time for a program
Instructioncount
Clock cycles perinstruction(CPI)
Clock cycletime

Units of measure
Seconds for the program
Instruction executed for the program
Average number of clock cycles per instruction
Seconds per clock cycle

CPU execution time for a program = CPU clock cycles for a program * Clock cycle time.

Factors affecting the CPU performance

The performance of a program depends on the algorithm, the language, the compiler, the architecture,andtheactualhardware.Thefollowinglistsummarizeshowthesecomponentsaffect the factors in the CPU performanceequation.

1. Algorithm –affects Instruction count, possiblyCPI

The algorithm determines the number of source program instructions executed and hence thenumberofprocessorinstructionsexecuted.ThealgorithmmayalsoaffecttheCPI,byfavoring slower or faster instructions. For example, if the algorithm uses more floating-point operations, it will tend to have a higherCPI.

2. Programming language - affects Instructioncount,CPI

The programming language certainly affects the instruction count, since statements in the languageareTranslatedtoprocessorinstructions,whichdetermineinstructioncount.Thelanguage may also affect the CPI because of its features; for example, a language with heavy support for data abstraction (e.g., Java) will require indirect calls, which will use higher-CPIinstructions.

3. Compiler - affects Instruction count,CPI
Theefficiencyofthecompileraffectsboththeinstructioncountandaveragecyclesperinstruction, since the compiler determines the translation of the source language instructions into computer instructions. The compiler's role can be very complex and affect the CPI in complexways.

4. Instruction set architecture - affects Instruction count, clock rate,CPI

The instruction set architecture affects all three aspects of CPU performance, since it affectstheinstructionsneededforafunction,thecostincyclesofeachinstruction,andtheoverall clock rate of theprocessor.

Amdahl's law

Amdahl'slawstatesthattheperformanceimprovementtobegainedfromusingsomefaster mode of execution is limited by the fraction of the time the faster mode can beused.

Speedup=Performanceforentiretaskusingtheenhancement/Performanceforentiretaskwithout using theenhancement

POWER WALL

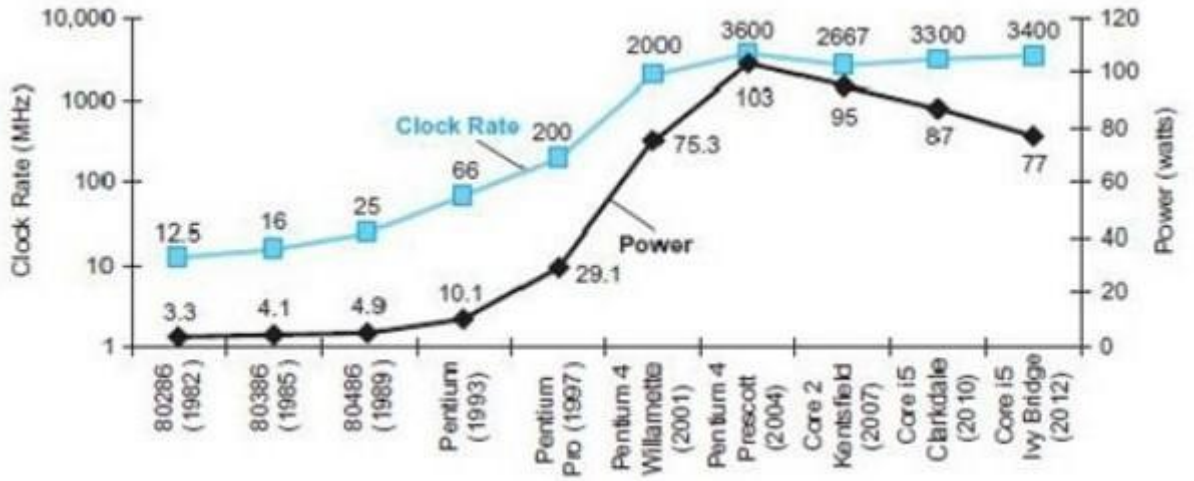


Fig.1.3 Clock rate and power for Intel x86 microprocessors

Fig.1.3 Clock rate and power for Intel x86 microprocessors

The dominant technology for integrated circuits is called CMOS (complementary metal oxidesemiconductor).ForCMOS,theprimarysourceofenergyconsumptionisso-calleddynamic energy—thatis,energythatisconsumedwhentransistorsswitchstatesfrom0to1andviceversa. The dynamic energy depends on the capacitive loading of each transistor and the voltageapplied.

UNIPROCESSORS TO MULTIPROCESSORS

Uniprocessor :

- A type of architecture that is based on a single computing unit. All operations (additions, multiplications, etc) are done sequentially on theunit.

Multiprocessor :

- Atypeofarchitecturethatisbasedonmultiplecomputingunits.Someoftheoperations(notall, mind you) are done in parallel and the results are joinedafterwards.

There are many types of classifications for multiprocessor architectures, the most commonly known would be the Flynn Taxonomy.

MIPS(originallyanacronymforMicroprocessorwithoutInterlockedPipelineStages)isareduced instruction set computer (RISC) instruction set architecture (ISA) developed by MIPS Technologies.

Thepowerlimithasforcedadramaticchangeinthedesignofmicroprocessors.Since2002, theratehasslowedfromafactorof1.5per yeartoafactorof1.2per year.Ratherthancontinuing to decrease the response time of a single program running on the single processor, as of 2006 all desktop and server companies are shipping microprocessors with multiple processors per chip, wherethebenefitisoftenmoreonthroughputthanonresponsetime.Toreduceconfusionbetween the words processor and microprocessor, companies refer to processors as "cores," and such microprocessors are generically called multicoremicroprocessors.

Hence, a "quadcore" microprocessor is a chip that contains four processors or four cores.Inthepast,programmerscouldrelyoninnovationsinhardware,architecture,andcompilers todoubleperformanceoftheirprogramsevery18monthswithouthavingtochangealineofcode. Today,forprogrammerstogetsignificantimprovementinresponsetime,theyneedtorewritetheir programstotakeadvantageofmultipleprocessors.Moreover,togetthehistoricbenefitofrunning faster on new microprocessors, programmers will have to continue to improve performance of their code as the number of coresincreases.

INSTRUCTIONS

Instruction format

The instruction format of an instruction is usually depicted in a rectangular box symbolizingthebitsoftheinstructionastheyappearinmemorywordsorinacontrolregister.An instruction format defines the layout of the bits of an instruction, in terms of its constituent parts. The bits of an instruction are divided into groups called fields. The most common fields found in instruction formatsare:

- An operation code field that specifies the operation to beperformed.
- An address field that designates a memory address or a processorregister.
- A mode field that specifies the way the operand or the effective address isdetermined

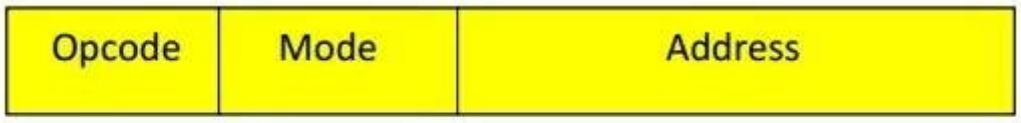


Fig 1.4 Instruction fields

Fig 1.4 Instruction fields

Other special fields are sometimes employed under certain circumstances. The operation codefieldofaninstructionisagroupofbitsthatdefinevariousprocessoroperations,suchasadd, subtract, complement and shift. Address fields contain either a memory address field or a register address. Mode fields offer a variety of ways in which an operand ischosen.

There are mainly four types of instruction formats:

- Three addressinstructions
- Two addressinstructions
- One addressinstructions
- Zero addressinstructions

Three address instructions

Computers with three address instructions formats can use each address field to specify eitheraprocessorregisteroramemoryoperand.Theprograminassemblylanguagethatevaluates X= (A+B)*(C+D) is shown below, together with comments that explain the register transfer operation of eachinstruction.

Add R1,A,B	R1←M [A] + M[B]
Add R2,C,D	R2←M [C] + M[D]
MulX,R1,R2	M [X]←R1 +R2

Itisassumedthatthecomputerhastwoprocessorregisters,R1andR2.ThesymbolM[A] denotestheoperandatmemoryaddresssymbolizedbyA.theadvantageofthethreeaddressformat is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that the binary coded instructions require too many bits to specify three addresses. An exampleofan commercial computer that uses three address instructions is the Cyber 170.The instruction formats in the Cyber computer are restricted to either three register address fields or two register address fields and one memory addressfield.

Two address instructions

Twoaddressinstructionsarethemostcommonincommercialcomputers.Hereagaineach address field can specify either a processor register or a memory word. The program to evaluate X= (A+B)*(C+D) is asfollows:

```
MOV   R1, A        R1←M [A]
ADD   R2, B        R1←R1 + M [B]
MOV   R2, C        R2←M [C]
ADD   R2, D        R2←R2 + M [D]
MUL   R1, R2       R1←R1 * R2
MOV   X, R1        M [X]←R1
```

The MOV instruction moves or transfers the operands to and from memory and processor registers.Thefirstsymbollistedinaninstructionisassumedtobebothasourceandthedestination where the result of the operation istransferred.

One address instructions

One address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division there is a need for a second register. However, here we will neglect the second register and assume that the AC contains the result of all operations. The program to evaluate X= (A+B)*(C+D) is

```
LOAD    A      AC←M [A]
ADD     B      AC←AC + M [B]
STORE   T      M [T]←AC
LOAD    C      AC←M [C]
ADD     D      AC←AC +M [D]
MUL     T      AC←AC * M [T]
STORE   X      M [X]←AC
```

All operations are done between the AC register and a memory operand. T is the address of a temporary memory location required for storing the intermediate result. Commercially available computers also use this type of instruction format.

Zero address instructions

AstackorganizedcomputerdoesnotuseanaddressfieldfortheinstructionsADDandMUL.The PUSH and POP instructions, however, need an address field to specify the operand that communicateswiththestack.ThefollowingprogramshowshowX=(A+B)*(C+D)willbewritten for a stack organized computer.(TOS stands for top ofstack.)

PUSH	A	TOS←A
PUSH	B	TOS←B
ADD		TOS←(A + B)
PUSH	C	TOS←C
PUSH	D	TOS←D
ADD		TOS←(C+D)
MUL		TOS←(C+D) * (A+B)
POP	X	M[X]←TOS

To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse polish notation. The name "zero address" is given to this type of computer because of the absence of an address field in computationalinstructions.

OPERATIONS AND OPERANDS

Everycomputermustbeabletoperformarithmetic.TheMIPSassemblylanguagenotation add a, b, c instructs a computer to add the two variables b and c and to put their sum in a. This notation is rigid in that each MIPS arithmetic instruction performs only one operation and must always have exactly three variables. For example, suppose we want to place the sum of four variables b, c, d, and e into variablea.

The following sequence of instructions adds the four variables:

```
add a, b, c # The sum of b and c is placed in a
        add a, a, d # The sum of b, c, and d is now in a
        add a, a, e # The sum of b, c, d, and e is now in a
```

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1, $s2, $s3	$s1 = $s2 + $s3	Three operands; data in registers
	subtract	sub $s1, $s2, $s3	$s1 = $s2 - $s3	Three operands; data in registers
	add immediate	addi $s1, $s2, 100	$s1 = $s2 + 100	Used to add constants
Data transfer	load word	lw $s1, 100($s2)	$s1 = Memory[$s2 + 100]	Word from memory to register
	store word	sw $s1, 100($s2)	Memory[$s2 + 100] = $s1	Word from register to memory
	load byte	lb $s1, 100($s2)	$s1 = Memory[$s2 + 100]	Byte from memory to register
	store byte	sb $s1, 100($s2)	Memory[$s2 + 100] = $s1	Byte from register to memory
	load upper immediate	lui $s1, 100	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Conditional branch	branch on equal	beq $s1, $s2, 25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1, $s2, 25	if ($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1, $s2, $s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for beq, bne
	set less than immediate	slti $s1, $s2, 100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare less than constant
Uncondi-tional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

Fig 1.5 MIPS Instructions

Thus, it takes three instructions to sum the four variables. The words to the right of the sharp symbol (#) on each line above are comments for the human reader, so the computerignores them.

MIPS operands

Name	Example	Comments
32 registers	$s0-$s7, $t0-$t9, $zero, $a0-$a3, $v0-$v1, $gp, $fp, $sp, $ra, $at	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register $zero always equals 0. Register $at is reserved for the assembler to handle large constants.
2^{30} memory words	Memory[0], Memory[4], ..., Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

Fig. 1.6 MIPS Operands

REPRESENTING INSTRUCTIONS

Numbersarekeptincomputerhardwareasaseriesofhighandlowelectronicsignals,andsothey are considered base 2 numbers. (Just as base 10 numbers are called decimal numbers, base 2 numbers are called binary numbers.) A single digit of a binary number is thus the "atom" of computing,sinceallinformationiscomposedofbinarydigitsorbits.Thisfundamentalbuilding

blockcanbeoneoftwovalues,whichcanbethoughtofasseveralalternatives:highorlow,onor off, true or false, or 1 or0.

Instructionsarealsokeptinthecomputerasaseriesofhighandlowelectronicsignalsand may be represented as numbers. In fact, each piece of an instruction can be considered as an individual number, and placing these numbers side by side forms theinstruction.

Sinceregistersarepartofalmostallinstructions,theremustbeaconventiontomapregister namesintonumbers.InMIPSassemblylanguage,registers$s0to$s7mapontoregisters16to23, andregisters$t0to$t7mapontoregisters8to15.Hence,$s0meansregister16,$s1meansregister 17, $s2 means register 18, . . . , $t0 means register 8, $t1. means register 9, and soon.

MIPS Fields

MIPS fields are given names to make them easier to discuss.

The meaning of each name of the fields in MIPS instructions:

op: Basic operation of the instruction, traditionally called the opcode. rs: The first register source operand.
rt: The second register source operand.
rd: The register destination operand. It gets the result of the operation.
shamt: Shift amount.

funct: Function. This field selects the specific variant of the operation in the op field and is sometimes called the function code.

A problem occurs when an instruction needs longer fields than those specified above. Forexample, the load word instruction must specify two registers and a constant. If the addressweretouseoneofthe5-bitfieldsintheformatabove,theconstantwithintheload wordinstructionwouldbelimitedtoonly25or32.Thisconstantisusedtoselectelements fromarraysordatastructures,anditoftenneedstobemuchlargerthan32.This5-bitfield is too small to beuseful.

This leads us to the hardware design principle: Good design demands good compromises.

Today’s computers are built on two key principles:

1. Instructions are represented asnumbers.
2. Programs are stored in memory to be read or written, just likenumbers.

These principles lead to the stored-program concept; its invention let the computing genie out of its bottle. specifically, memory can contain the source code for an editor program, the corresponding compiled machine code, the text that the compiled program is using, andeven

thecompilerthatgeneratedthemachinecode.Oneconsequenceofinstructionsasnumbersisthat programs are often shipped as files of binary numbers. The commercial implication is that computers can inherit ready-made software provided they are compatible with an existing instruction set. Such “binary compatibility” often leads industry to align around a small number of instruction setarchitectures.

LOGICAL OPERATIONS

Although the first computers operated on full words, it soon became clear that it was useful to operate on fields of bits within a word or even on individual bits. Examining characters within a word, each of which is stored as 8 bits, is one example of such an operation. It follows that operations were added to programming languages and instruction set architectures to simplify,amongotherthings,thepackingandunpackingofbitsintowords. Theseinstructionsare called logicaloperations.

CONTROL OPERATIONS

What distinguishes a computer from a simple calculator is its ability to make decisions. Based on the input data and the values created during computation, different instructions execute. Decision making is commonly represented in programming languages using the if statement, sometimes combined with go to statements and labels. MIPS assembly language includes two decision-making instructions, similar to an if statement with a go to. The first instruction is

beq register1, register2, L1

This instruction means go to the statement labeled L1 if the value in register1 equals the value in register2. The mnemonic beq stands for branch if equal.

The second instruction is

bne register1, register2, L1

It means go to the statement labeled L1 if the value in register1 does not equal the value in register2.Themnemonicbnestandsforbranchifnotequal.Thesetwoinstructionsaretraditionally called conditionalbranches.

Case/Switch statement

Mostprogramminglanguageshaveacaseorswitchstatementthatallowstheprogrammer to select one of many alternatives depending on a single value. The simplest way to implement switch is via a sequence of conditional tests, turning the switch statement into achain of if-then-else statements.

Sometimes the alternatives may be more efficiently encoded as a table of addresses of alternative instruction sequences, called a jump address table, and the program needs only to indexintothetableandthenjumptotheappropriatesequence.Thejumptableisthenjustanarray of words containing addresses that correspond to labels in thecode.

To support such situations, computers like MIPS include a jump register instruction (jr), meaning an unconditional jump to the address specified in a register. The program loads the appropriate entry from the jump table into a register, and then it jumps to the proper address using a jump register.

ADDRESSING AND ADDRESSING MODES

To perform any operation, the corresponding instruction is to be given to the microprocessor. In each instruction, programmer has to specify 3 things:

- Operation to beperformed.
- Address of source ofdata.
- Address of destination ofresult.

Definition:

- The different ways in which the location of an operand is specified in an instruction are referred to as addressingmodes.

· The method by which the address of source of data or the address of destination of result is given in the instruction is called AddressingModes

· Computersuseaddressingmodetechniquesforthepurposeofaccommodatingoneorboth of the followingprovisions:

- To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and programrelocation.

- To reduce the number of bits in the addressing field of theinstruction.

IMPLEMENTATION OF VARIABLES AND CONSTANTS

Variablesandconstantsarethesimplestdatatypesandarefoundinalmosteverycomputer program. A variable is represented by allocating a register or a memory location to hold its value. Thus, the value can be changed as needed using appropriateinstructions.

1. Register addressing mode - The operand is the contents of a processor register; the name (address) of the register is given in theinstruction.

Example: **MOVE R1,R2**
This instruction copies the contents of register R2 to R1.

2. Absoluteaddressingmode-Theoperandisinamemorylocation;theaddressofthislocation is given explicitly in the instruction. (In some assembly languages, this mode is calledDirect.)

Example: **MOVE LOC,R2**
This instruction copies the contents of memory location of LOC to register R2.

3. Immediate addressing mode - The operand is given explicitly in the instruction.
Example: **MOVE #200 ,R0**
The above statement places the value 200 in the register R0. A common convention is to use the sharpsign(#)infrontofthevaluetoindicatethatthisvalueistobeusedasanimmediateoperand.

INDIRECTION AND POINTERS

Intheaddressingmodesthatfollow,theinstructiondoesnotgivetheoperandoritsaddress explicitly. Instead, it provides information from which the memory address of the operand can be determined. We refer to this address as the effective address (EA) of theoperand.

4. Indirect addressingmode

Theeffectiveaddressoftheoperandisthecontentsofaregisterormemorylocationwhose address appears in theinstruction.
Example **Add (R2),R0**

Register R2 is used as a pointer to the numbers in the list, and the operands are accessed indirectly through R2. The initialization section of the program loads the counter value n from memory location N into Rl and uses the Immediate addressing mode to place the address value NUM 1, which is the address of the first number in the list, into R2.

INDEXING AND ARRAY

It is useful in dealing with lists and arrays.

5. Indexmode

Theeffectiveaddressoftheoperandisgeneratedbyaddingaconstantvaluetothecontents of a register. The register used may be either a special register provided for this purpose, or,more commonly; it may be anyone of a set of general-purpose registers in the processor. In either case, it is referred to as an index register. We indicate the Index mode symbolicallyas

X(Ri).

Where X denotes the constant value contained in the instruction and Ri is the name of the registerinvolved.Theeffectiveaddressoftheoperandisgivenby**EA=X+[Ri].**Thecontentsof the index register are not changed in the process of generating the effectiveaddress.

RELATIVE ADDRESSING

An useful version of this mode is obtained if the program counter, PC, is used instead ofa general purpose register. Then, X(PC) can be used to address a memory location that is X bytes away from the location presently pointed to by the program counter. Since the addressed location is identified "relative" to the program counter, which always identifies the current executionpoint in a program, the name Relative mode is associated with this type ofaddressing.

6.Relative mode - The effective address is determined by the Index mode using the programcounterinplaceofthe general-purposeregisterRi.Thismodecanbeusedtoaccessdata operands. But, its most common use is to specify the target address in branch instructions. An instructionsuchas**Branch>OLOOP**causesprogramexecutiontogotothebranchtargetlocation identified by the name LOOP if the branch condition is satisfied. This location can be computed byspecifyingitasanoffsetfromthecurrentvalueoftheprogramcounter.Sincethebranchtarget may be either before or after the branch instruction, the offset is given as a signednumber.

ADDITIONAL MODES

The two additional modes described are useful for accessing data items in successive locations in the memory.

7. Autoincrement mode - The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list. We denote the Autoincrement mode by putting the specified register in parentheses, to show that the contents of the register are used as the effective address, followed by a plus sign to indicate that these contents are tobe

incremented after the operand is accessed. Thus, the Autoincrement mode is written as **(Ri) +.** As a companion for the Autoincrement mode, another useful mode accesses the items of a list in the reverse order:

8. Autodecrement mode - The contents of a register specified in the instruction is firstautomaticallydecrementedandisthenusedastheeffectiveaddressoftheoperand.Wedenote the Autodecrement mode by putting the specified register in parentheses, preceded by a sign to indicate that the contents of the register are to be decremented before being used as the effective address. Thus, we write **-(Ri)**

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	Ri	EA = Ri
Absolute (Direct)	LOC	EA = LOC
Indirect	(Ri)	EA = [Ri]
	(LOC)	EA = [LOC]
Index	X(Ri)	EA = [Ri] + X
Base with index	(Ri,Rj)	EA = [Ril + [Rj]
Base with index and offset	X(Ri,Rj)	EA = [Ri] + [Rj] + X
Relative	X(PC)	EA = [PC] +X
Autoincrement	(Ri)+	EA = [Ri]; Increment Ri
Autodecrement	-(Ri)	Decrement Ri; EA = [Ri]

EA = effective address Value = a signed number

Fig. 1.7 Addressing modes

Fig. 1.7 Addressing modes

Illustration of Addressing Modes

- Implied addressingmode
- Immediate addressingmode
- Direct addressingmode
- Indirect addressingmode
- Register addressingmode

- Register Indirect addressingmode
- Autoincrement or Autodecrementaddressingmode
- Relative addressingmode
- Indexed addressingmode
- Base register addressingmode
- **Implied addressingmode**

In this mode the operands are specified implicitly in the definition of the instruction. Forexample the 'complement accumulator' instruction is an implied mode instruction because the operand in the accumulator register is implied in the definition of the instruction itself. All register reference instructions that use an accumulator are implied mode instructions. Zero address instructions in a stack organized computer are implied mode instructions since the operands are implied to be on the top of thestack.

Example:CMA

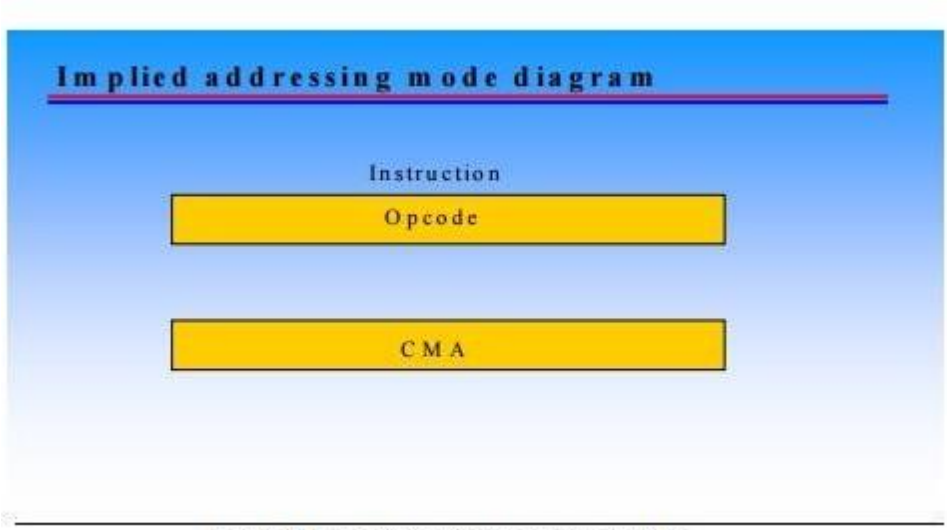


Fig1.8 Implied addressing mode

Fig1.8 Implied addressing mode

Immediate addressing mode

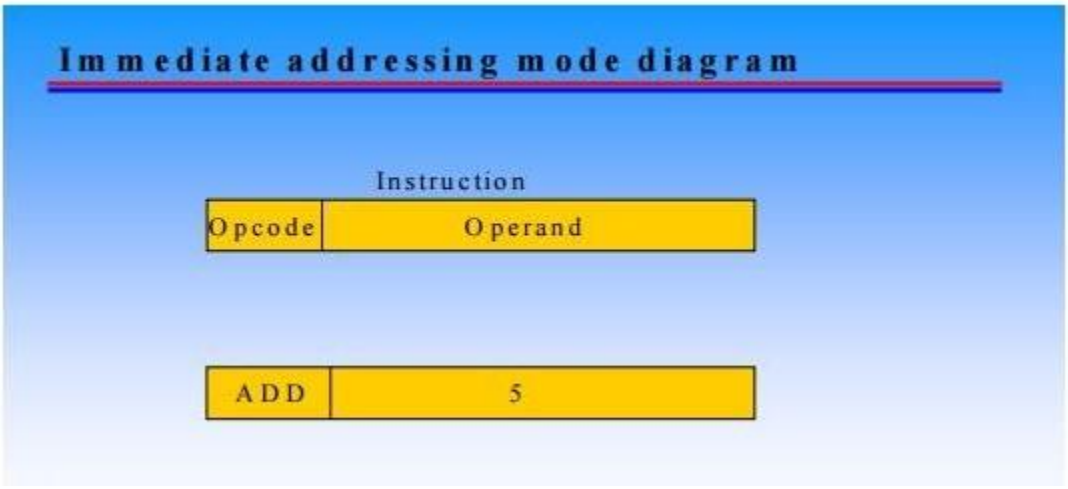


Fig. 1.9 Immediate addressing mode

In this mode the operand is specified in the instruction itself. In other words, an immediate mode instruction has a operand field rather than an address field. The operand field contains the actual operandtobeusedinconjunctionwiththeoperationspecifiedintheinstruction.Immediatemode instructions are useful for initializing registers to a constantvalue.
Example: ADD 5

- Add 5 to contents accumulatorof
- 5 is operand

Advantages anddisadvantages

- No memory reference to fetchdata
- Fast
- Limitedrange

Direct addressingmode

Inthismodetheeffectiveaddressisequaltotheaddresspartoftheinstruction.Theoperandresides in memory and its address is given directly by the address field of instruction. In a branch type instruction the address field specifies the actual branchaddress

Effective address (EA) = address field (A)

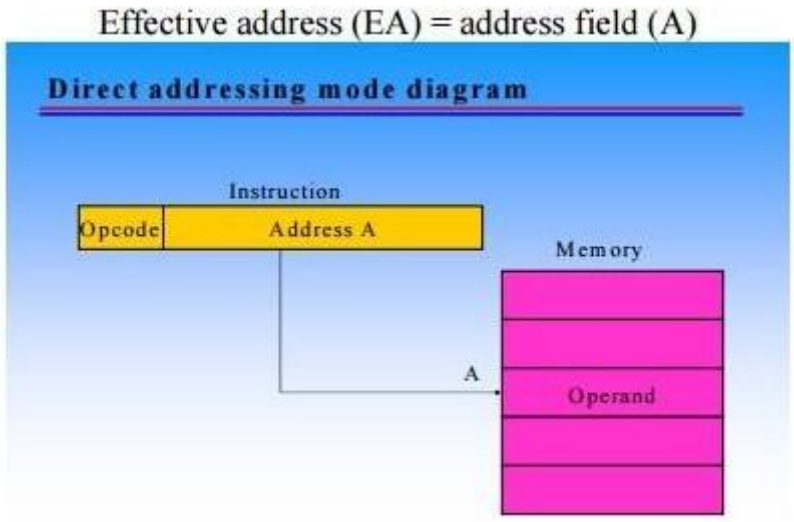


Fig. 1.10 Direct addressing mode

e.g. LDAA

Look in memory at address A for operand.
Load contents of A to accumulator

Advantages and disadvantages

- Single memory reference to accessdata
- No additional calculations to work out effectiveaddress
- Limited addressspace

Indirect addressing mode

In this mode the address field of the instruction gives the address where the effective address is stored in memory/register. Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.

EA = address contained in register/memory location

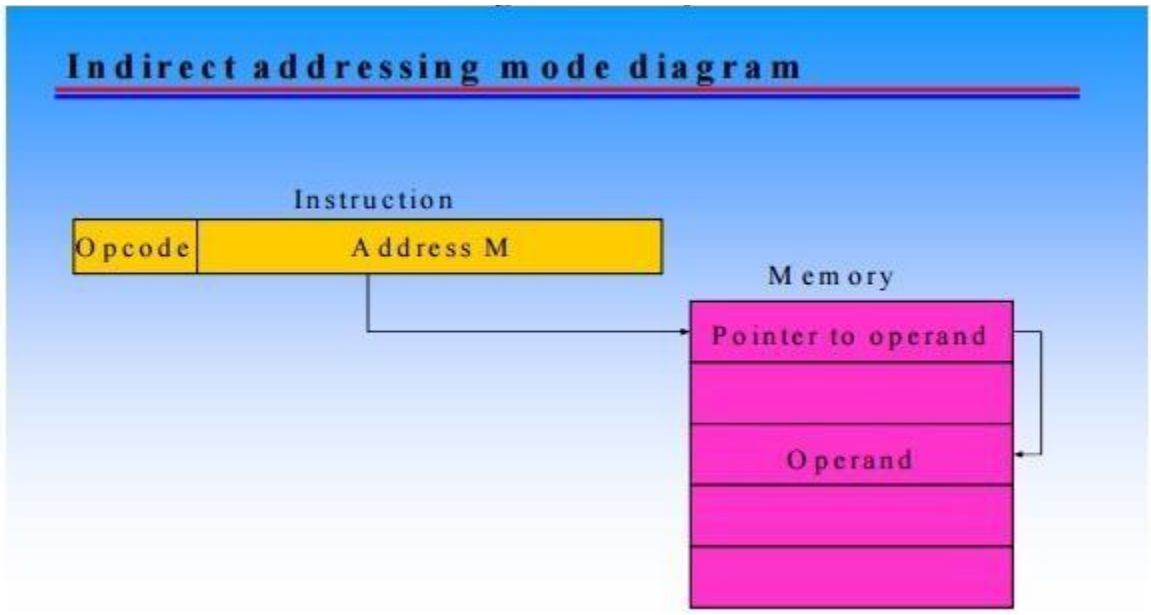


Fig. 1.11 Indirect addressing mode

Example **Add (M)**

- **Look in M, find address contained in M and look there foroperand**
- **Add contents of memory location pointed to by contents of M toaccumulator.**

Register addressing mode

In this mode the operands are in the registers that reside within the CPU.
EA = R
Example : ADD R1,R2

Advantages and disadvantages

- No memory access. So very fastexecution.
- Very small address fieldneeded.
- Shorterinstructions
- Faster instructionfetch

- Limited number ofregisters.
- Multiple registers helpsperformance
- Requires good assembly programming or compilerwriting

Register indirect addressing mode

In this mode the instruction specifies a register in the CPU whose contents give the effective address of

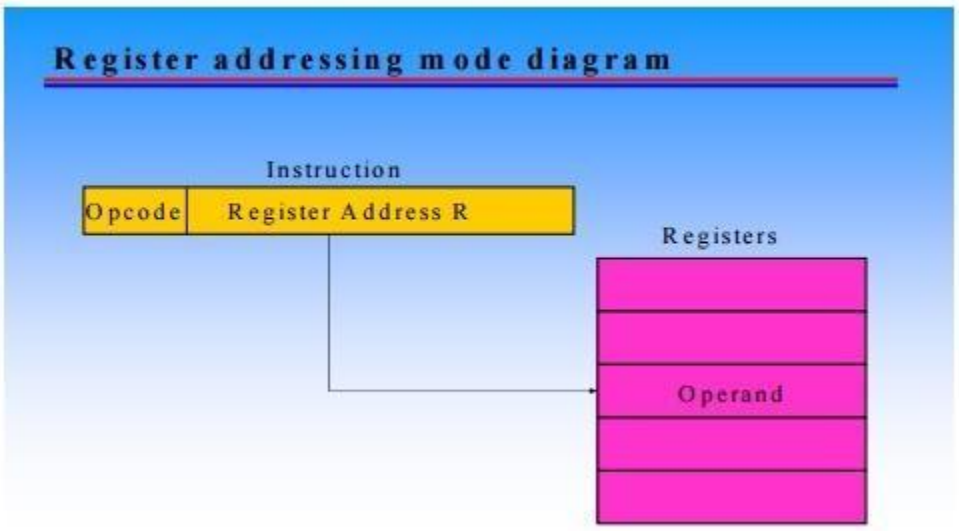


Fig.1.12 Register addressing mode

theoperandinthememory.Inotherwords,theselectedregistercontainstheaddressoftheoperand rather than the operand itself. Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previousinstruction.Theadvantageofaregisterindirectmodeinstructionisthattheaddressfield of the instruction uses fewer bits to select a register than would have been required to specify a memory addressdirectly.

Therefore EA = the address stored in the register R

- Operand is in memory cell pointed to by contents ofregister
- Example **Add(R2),R0**

Advantage

- Less number of bits are required to specify theregister.

- One fewer memory access than indirectaddressing.

Register Indirect addressing mode diagram

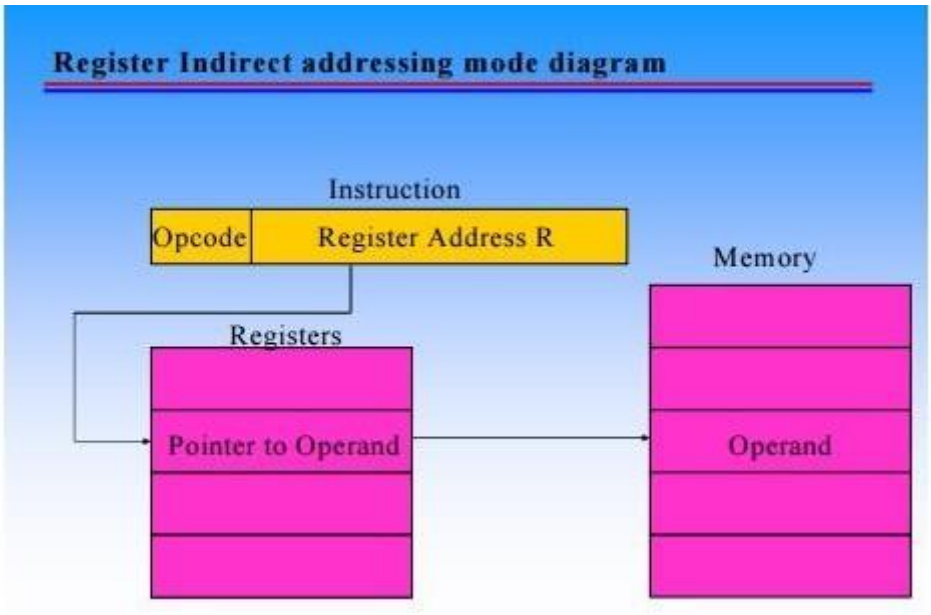


Fig. 1.13 Indirect addressing mode

Autoincrement or autodecrement addressing mode

Autoincrementmode-Theeffectiveaddressoftheoperandisthecontentsofaregisterspecified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next item in alist.

- We denote the Autoincrement mode by putting the specified register in parentheses, to show that the contents of the register are used as the effective address, followed by a plus sign to indicate that these contents are to be incremented after the operand is accessed. Thus, the Autoincrement mode is written as **(Ri) +**

Autodecrementmode-Thecontentsofaregisterspecifiedintheinstructionisfirstautomatically decremented and is then used as the effective address of theoperand.

We denote the Autodecrement mode by putting the specified register in parentheses, preceded by a

minus sign to indicate that the contents of the register are to be decremented before being used as the effective address. Thus, we write **- (Ri)**

- These two modes are useful when we want to access a table ofdata.
  ADD(R1)+

  will increment the register R1.
  LDA -(R1)

will decrement the register R1.

Relative addressing mode

In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address. Effective address is defined as the memory addressobtainedfromthecomputationdictatedbythegivenaddressingmode.Theaddresspartof the instruction is usually a signed number (in 2's complement representation) which can be either positive or negative. When this number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction.

Relativeaddressingisoftenusedwithbranchtypeinstructionswhenthebranchaddressis in the area surrounding the instruction word itself. It results in a shorter address field in the instruction format since the relative address can be specified with a smaller number of bits compared to the bits required to designate the entire memoryaddress.

EA = A + contents of PC

Example: PC contains 825 and address pa rt of instruction contains 24.

Aftertheinstructionisreadfromlocation825,thePCisincrementedto826.SoEA=826+24=850. The operand will be found at location 850 i.e. 24 memory locations forward from the address of the nextinstruction.

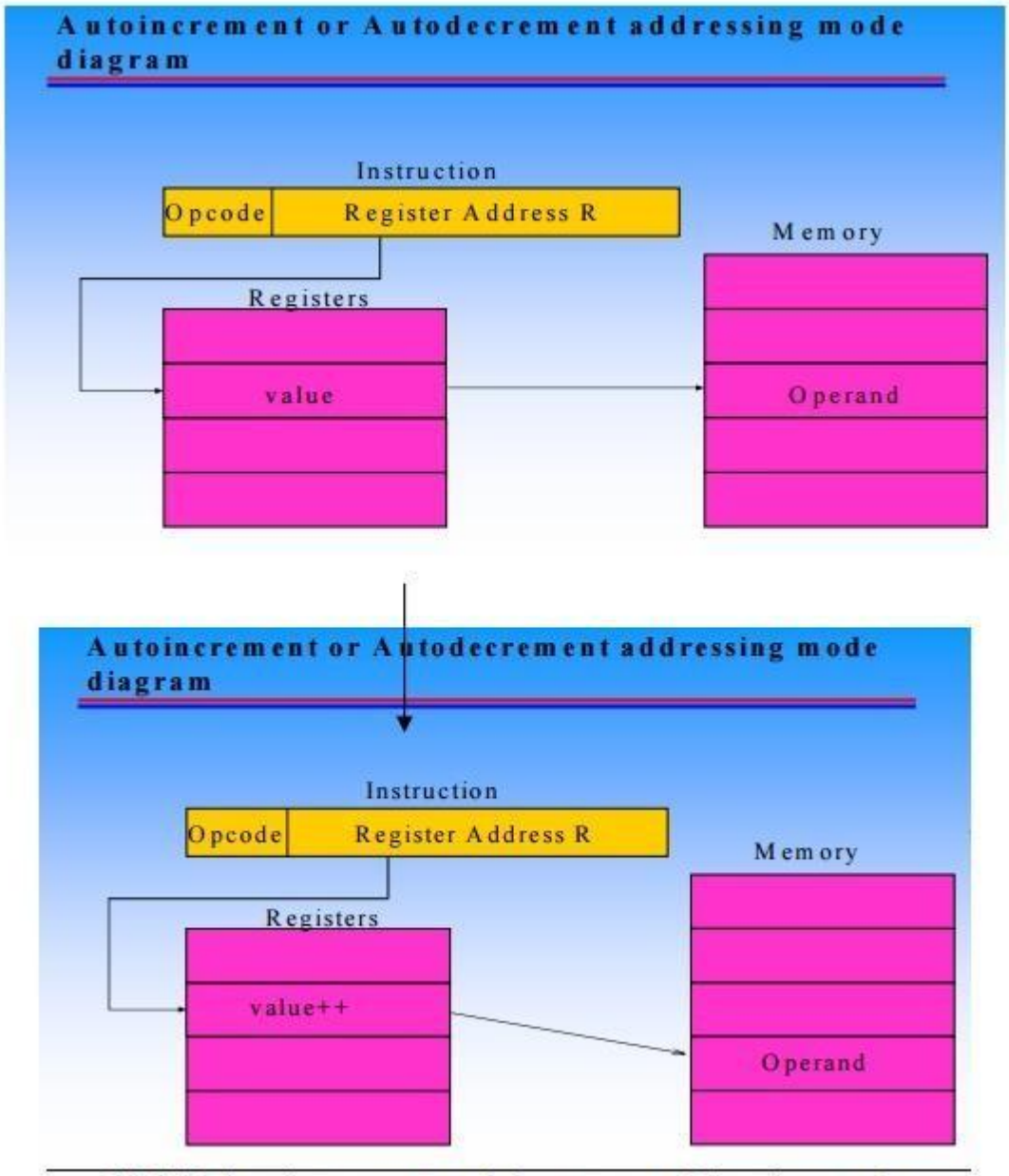


Fig1.14 Autoincrement and decrement addressing mode

Fig1.14 Autoincrement and decrement addressing mode

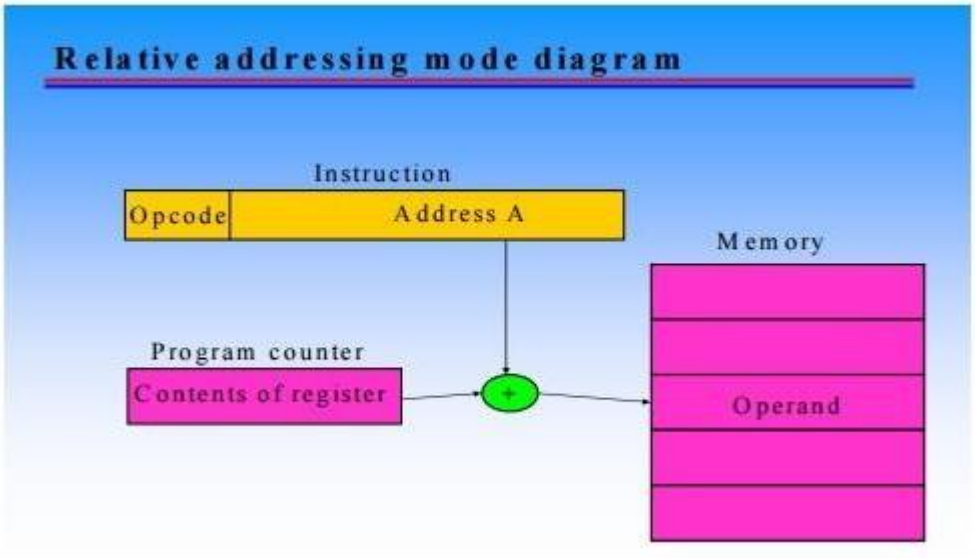


Fig.1.15 Relative addressing mode

Indexed addressing mode

Inthismodethecontentofanindexregisterisaddedtotheaddresspartoftheinstructiontoobtain theeffectiveaddress.TheindexregisterisaspecialCPUregisterthatcontainsanindexvalue.The address field of the instruction defines the beginning address of a data array in memory. Each operand in the array is store din memory relative to the beginning address. The distance between thebeginningaddressandtheaddressoftheoperandistheindexvaluestoredintheindexregister. Anyoperandinthearraycanbeaccessedwiththesameinstructionprovidedthattheindexregister contains the correct index value. The index register can be incremented to facilitate access to consecutiveoperands.Notethatifanindextypeinstructiondoesnotincludeanaddressfieldinits format, then the instruction converts to the register indirect mode ofoperation.

- Therefore EA = A + IR

- Example MOV AL , DS: disp [SI]Advantage
- Good for accessingarrays.

Base register addressing mode

Inthismodethecontentofbaseregisterisaddedtotheaddresspartoftheinstructiontoobtainthe effective address. This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register. The difference between the two modes is in the way they are used rather than in the way that they arecomputed.

An index register is assumed to hold an index number that is relative to the address part of the instruction.Abaseregisterisassumedtoholdabaseaddressandtheaddressfieldoftheinstruction gives a displacement relative to this base address. The base register addressing mode is used in computers to facilitate the relocation of the programs in memory. When programs and data are moved from one segment of memory to another, as required in multiprogramming systems, the address values of instructions must reflect this change of position. With abase
register, the displacement values of instructions do not have to change. Only the value of thebase register requires updating to reflect the beginning of a new memorysegment.

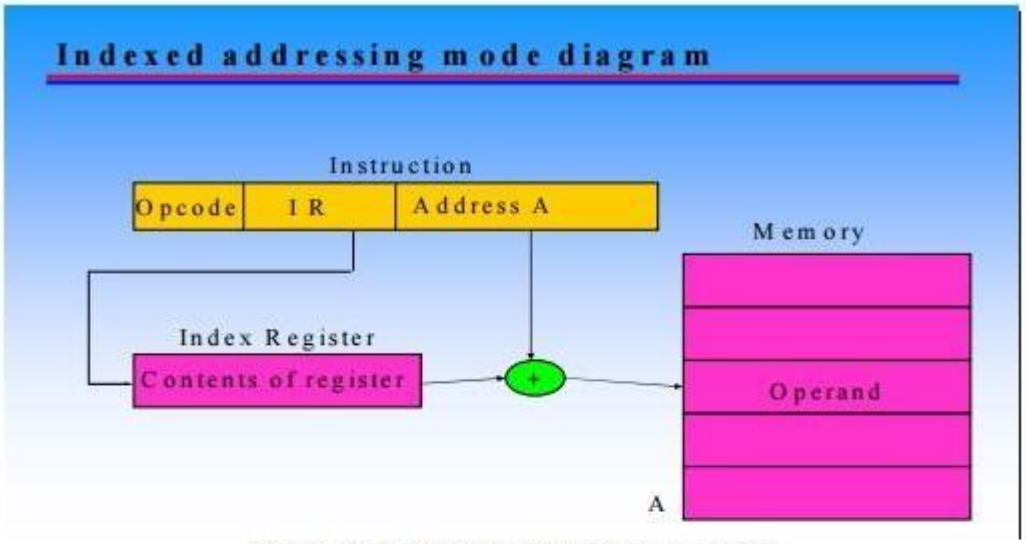


Fig.1.16 Indexed addressing mode

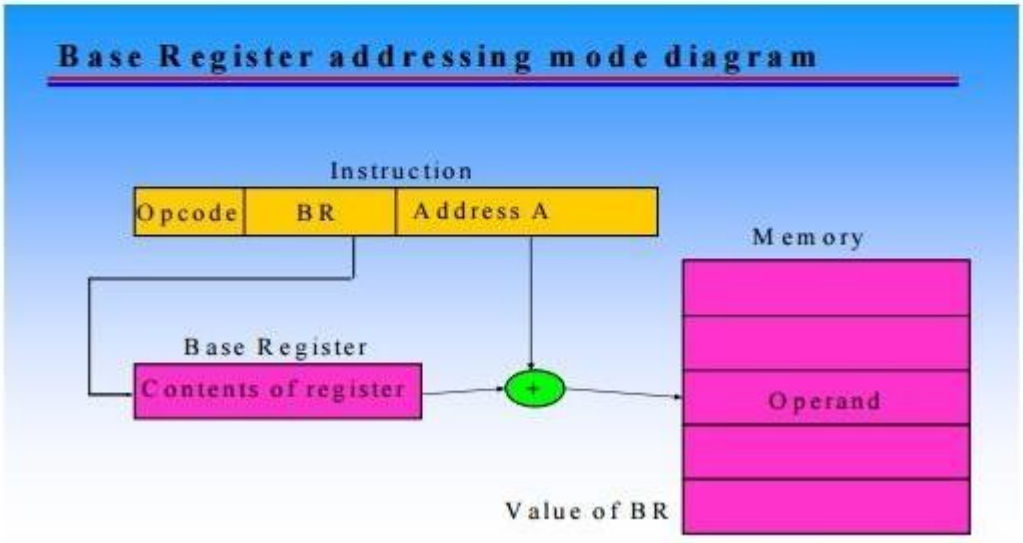


Fig.1.17 Base register addressing mode

• Therefore EA= A +BR
• For example: MOV AL, disp[BX]

Segment registers in 8086

MIPS Addressing Mode Summary

1. Immediate addressing, where the operand is a constant within the instructionitself
2. Register addressing, where the operand is aregister

3. Base or displacement addressing, where the operand is at the memory location whose address is the sum of a register and a constant in theinstruction

4. PC-relative addressing, where the branch address is the sum of the PC and a constant in the instruction

5. Pseudodirect addressing, where the jump address is the 26 bits of the instruction concatenated with the upper bits of thePC

UNIT II ARITHMETIC OPERATIONS

Most computer operations are executed in the arithmetic and logic unit (ALU) of the processor. Consider a typical example: Suppose two numbers located in the memory are to be added. Theyarebroughtintotheprocessor, andtheactualadditioniscarriedoutbytheALU.The sum may then be stored in the memory or retained in the processor for immediateuse.

Any other arithmetic or logic operation, for example, multiplication, division, or comparison of numbers, is initiated by bringing the required operands into the processor, where the operation is performed by the ALU. When operands are brought into the processor, they are stored in high-speed storage elements called registers. Each register can store one word of data. Access times to registers are somewhat faster than access times to the fastest cache unit in the memory hierarchy.

The control and the arithmetic and logic units are many times faster than other devices connected to a computer system.

ADDITION AND SUBTRACTION

Digitsareaddedbitbybitfrom righto left,withcarriespassedtothenextdigittotheleft. Subtractionusesaddition:Theappropriateoperandissimplynegatedbeforebeingadded.overflow occurswhentheresultfromanoperationcannotberepresentedwiththeavailablehardware.When

adding operands with different signs, overflow cannot occur. The reason is the sum must be no larger than one of the operands. For example, –10 + 4 = –6. Since the operands fit in 32 bits and the sum is no larger than an operand, the sum must fit in 32 bits as well.

Therefore no overflow can occur when adding positive and negative operands. There are similar restrictions to the occurrence of overflow during subtract, but it's just the opposite principle: When the signs of the operands are the same, overflow cannot occur. Overflow occurs in subtraction when a negative number is subtracted from a positive number and get a negative result, or when a positive number is subtracted from a negative number and get a positive result. Thismeansaborrowoccurredfromthesignbit.Unsignedintegersarecommonlyusedformemory addresses where overflows areignored.

Thecomputerdesignermustthereforeprovideawaytoignoreoverflowinsomecasesand to recognize it in others. The MIPS solution is to have two kinds of arithmetic instructions to recognize the twochoices:

■ Add (add), add immediate (addi), and subtract (sub) cause exceptions onoverflow.
■ Add unsigned (addu), add immediate unsigned (addiu), and subtract unsigned (subu) donot

cause exceptions on overflow.

Because C ignores overflows, the MIPS C compilers will always generate the unsigned versions of the arithmetic instructions addu, addiu, and subu no matter what the type of the

variables. The MIPS Fortran compilers, however, pick the appropriate arithmetic instructions, depending on the type of the operands.

Operation	Operand A	Operand B	Result indicating overflow
$A+B$	≥ 0	≥ 0	< 0
$A+B$	< 0	< 0	≥ 0
$A-B$	≥ 0	< 0	< 0
$A-B$	< 0	≥ 0	≥ 0

Fig. 2.1 Overflow conditions for addition and subtraction.

The computer designer must decide how to handle arithmetic overflows. Although some languageslikeCignoreintegeroverflow,languageslikeAdaandFortranrequirethattheprogram be notified. The programmer or the programming environment must then decide what to dowhenoverflowoccurs.

MIPS detects overflow with an exception, also called an interrupt on many computers.An exception or interrupt is essentially an unscheduled procedure call. The address of the instruction that overflowed is saved in a register, and the computer jumps to a predefined address to invoke the appropriate routine for that exception. The interrupted address is saved so that in some situations the program can continue after corrective code is executed. MIPS includes a register called the exception program counter (EPC) to contain the address of the instruction that caused the exception. The instruction move from system control (mfc0) is used to copy EPC into a general-purpose register so that MIPS software has the option of returning to the offending instruction via a jump registerinstruction.

Addition and Subtraction Example

adding 6 to 7 in binary and then subtracting 6 from 7 in binary: 0000 0000 0000 0000 0000 0000 0000 0111two = 7

\+ 0000 0000 0000 0000 0000 0000 0000 0110two =

= 0000 0000 0000 0000 0000 0000 0000 1101two =13

Subtracting 6 from 7 can be done directly:

0000 0000 0000 0000 0000 0000 0000 0111two = 7

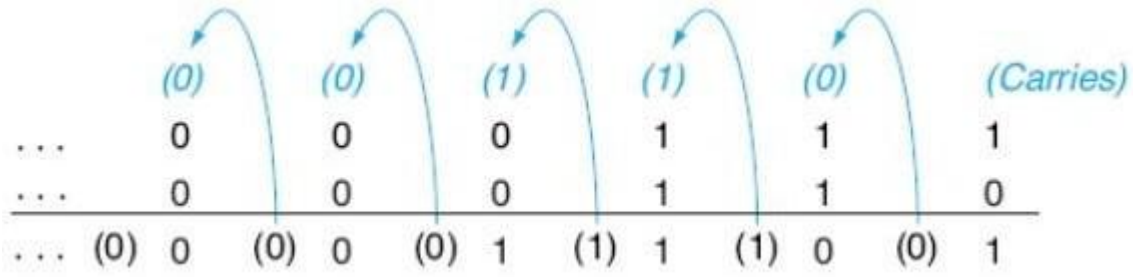
– 0000 0000 0000 0000 0000 0000 0000 0110two =6

= 0000 0000 0000 0000 0000 0000 0000 0001two =1

or via addition using the two's complement representation of –6: 0000 0000 0000 0000 0000 0000 0000 0111two = 7

\+ 1111 1111 1111 1111 1111 1111 1111 1010two =–6

= 0000 0000 0000 0000 0000 0000 0000 0001two =1



Instructions available

Add, subtract, add immediate, add unsigned, subtract unsigned.

Carry-Look Ahead Adder

• Binary addition would seem to be dramatically slower for large registers consider 0111 +0011
carries propagateleft-to-right
So 64-bit addition would be 8 times slower than 8- bit addition

• It is possible to build a circuit called a “carry look-ahead adder” that speeds up addition by eliminating the need to “ripple” carries through theword.

• Carry look-ahead isexpensive
• If n is the number of bits in a ripple adder, the circuit complexity (number of gates) isO(n)
• For full carry look-ahead, the complexity is O(n3)
• Complexity can be reduced by rippling smaller look-aheads: e.g., each 16 bit group ishandled

by four 4-bit adders and the 16-bit adders are rippled into a 64-bitadder

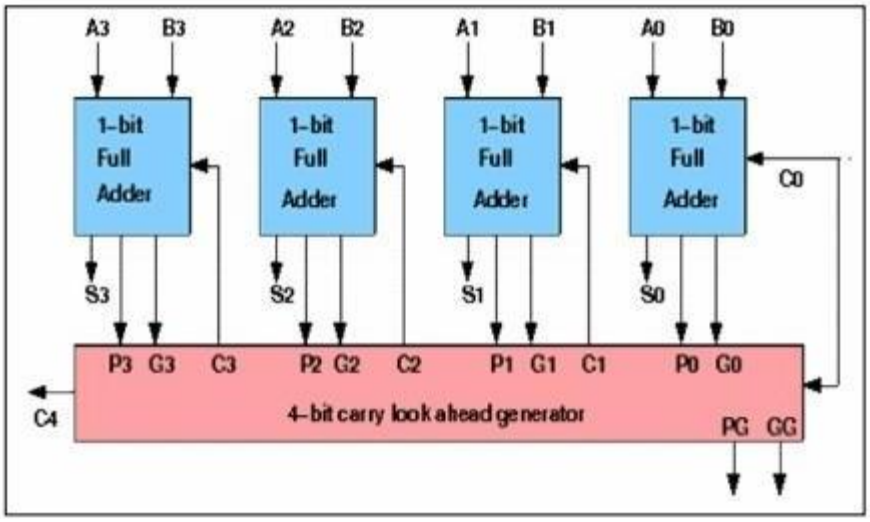


Fig. 2.2 Carry-look ahead adder

The advantage of the CLA scheme used in this circuit is its simplicity, because each CLA block calculates the generate and propagate signals for two bits only. This is much easier to understand than the more complex variants presented in other textbooks, where combinatoricallogicisusedtocalculatetheGandPsignalsoffourormorebits,andtheresultingadderstructure is slightly faster but also lessregular.

MULTIPLICATION

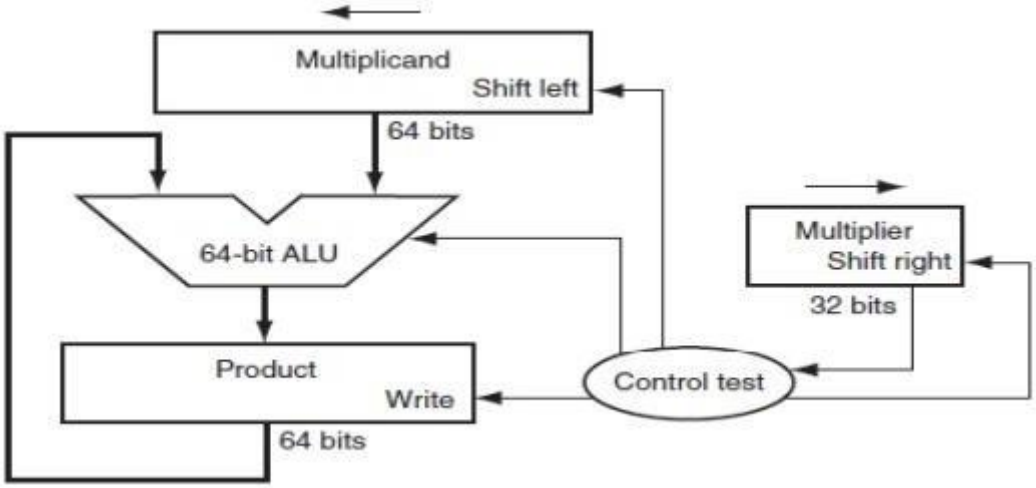


Fig. 2.3 First version of the multiplication hardware

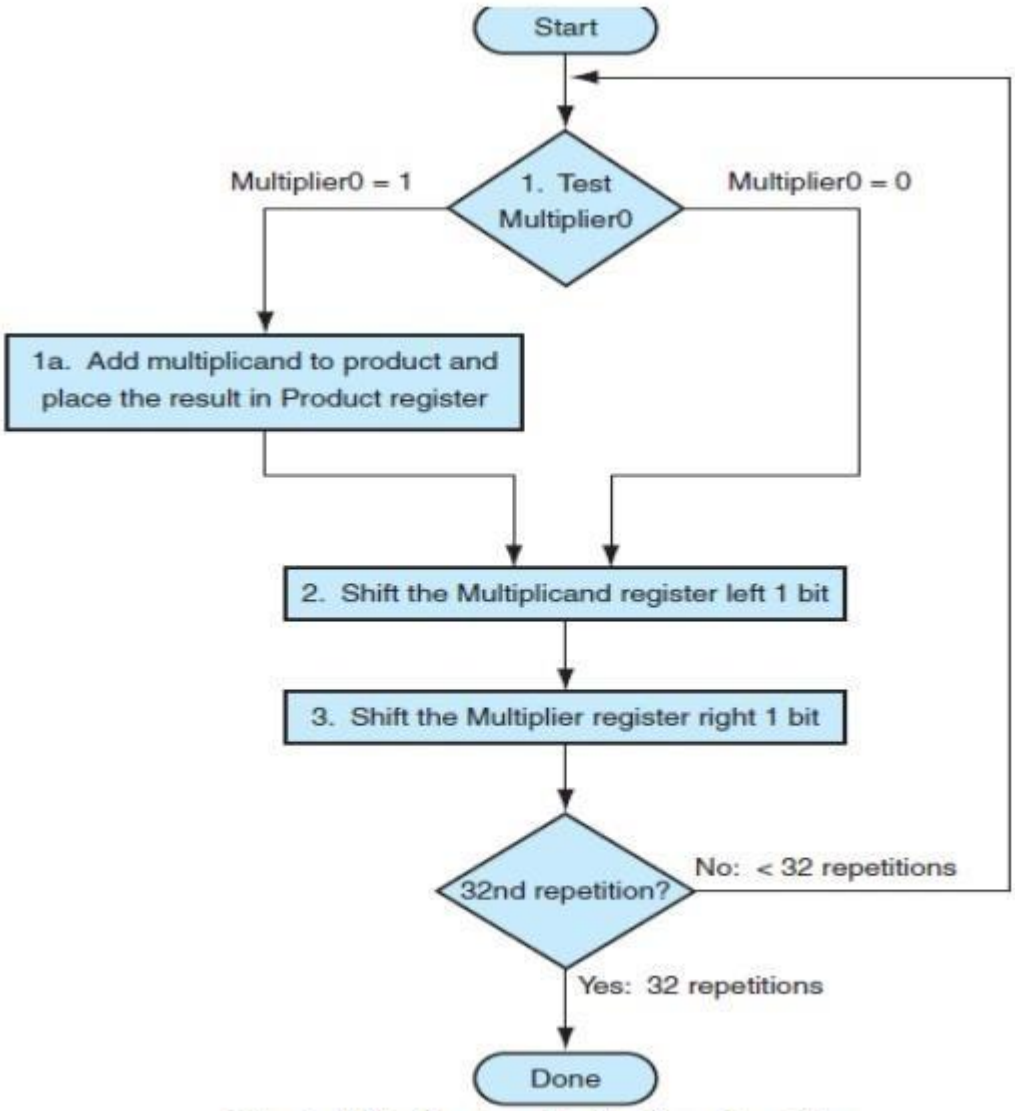


Fig. 2.4 The first multiplication algorithm

Multiplying 1000ten by 1001ten:

Multiplicand		1000
Multiplier	x	1001
		1000
		0000
		0000
		1000
Product		1001000

The first operand is called the multiplicand and the second the multiplier. The final result is called the product. As you may recall, the algorithm learned in grammar school is to take the digits of the multiplier one at a time from right to left, multiplying the multiplicand by the single digit of the multiplier and shifting the intermediate product one digit to the left of the earlier intermediate products.

The first observation is that the number of digits in the product is considerably larger than thenumberineitherthemultiplicandorthemultiplier.Infact,ifweignorethesignbits,thelength ofthemultiplicationofann-bitmultiplicandandanm-bitmultiplierisaproductthatisn+mbits long. That is, n + m bits are required to represent all possible products. Hence, like add, multiply must cope with overflow because we frequently want a 32-bit product as the result ofmultiplying two 32-bitnumbers.

In this example we restricted the decimal digits to 0 and 1. With only two choices, each step of the multiplication is simple:
1. Just place a copy of the multiplicand in the proper place if the multiplier digit is a 1,or
2. Place 0 (0 ¥ multiplicand) in the proper place if the digit is0.

The multiplier is in the 32-bit Multiplier register and that the 64-bit Product register is initializedto0.Over32stepsa32-bitmultiplicandwouldmove32bitstotheleft.Henceweneed a 64-bit Multiplicand register, initialized with the 32-bit multiplicand in the right half and 0 inthe left half. This register is then shifted left 1 bit each step to align the multiplicand with the sum being accumulated in the 64-bit Productregister.

Moore's Law has provided so much more in resources that hardware designers can now build a much faster multiplication hardware. Whether the multiplicand is to be added or not is known at the beginning of the multiplication by looking at each of the 32 multiplier bits. Faster multiplicationsarepossiblebyessentiallyprovidingone32-bitadderforeachbitofthemultiplier: one input is the multiplicand ANDed with a multiplier bit and the other is the output of a prior adder.

SIGNED MULTIPLICATION

- Inthesignedmultiplication,convertthemultiplierandmultiplicandtopositivenumbers and then remember the originalsigns.

- Thealgorithmsshouldthenberunfor31iterations,leavingthesignsoutofthecalculation
- The shifting steps would need to extend the sign of the product for signed numbers. When the algorithm completes, the lower word would have the 32-bitproduct.

FASTER MULTIPLICATION

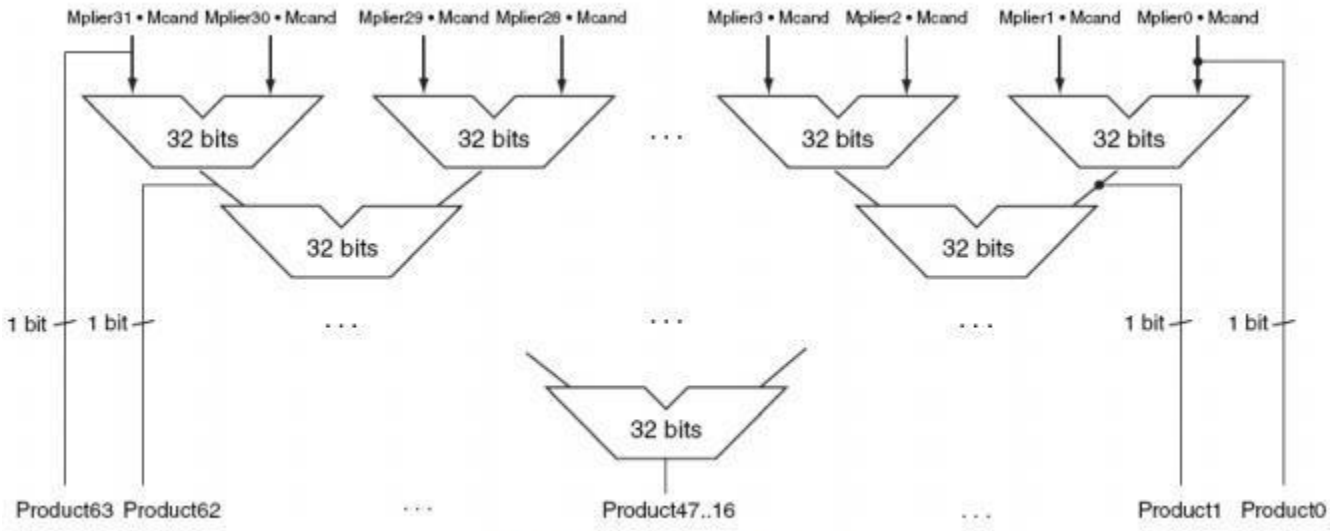


Fig.2.5 Faster multiplier

Fig.2.5 Faster multiplier

- Faster multiplications are possible by essentially providing one 32-bit adder for eachbit of the multiplier: one input is the multiplicand ANDed with a multiplier bit, and the other is the output of a prioradder.

- Connect the outputs of adders on the right to the inputs of adders on the left, making a stack of adders 32high.

- The above figure shows an alternative way to organize 32 additions in a parallel tree. Instead of waiting for 32 add times, we wait just the $\log_2(32)$ or five 32-bit addtimes.
- Multiply can go even faster than five add times because of the use of carry saveadders.

- It is easy to pipeline such a design to be able to support many multipliessimultaneously

Multiply in MIPS

MIPS provides a separate pair of 32-bit registers to contain the 64-bit product, called Hi and Lo. To produce a properly signed or unsigned product, MIPS has two instructions: multiply (mult) and multiply unsigned (multu). To fetch the integer 32-bit product, the programmer uses movefromlo(mflo).TheMIPSassemblergeneratesapseudoinstructionformultiplythatspecifies three generalpurpose registers, generating mflo and mfhi instructions to place the product into registers.

Booth Algorithm

Booth's Algorithm Registers and Setup

• 3 n bit registers, 1 bit register logically to the right of Q (denoted as Q-1)

• Register setup

— Q register <-multiplier

— Q-1 <- 0

— M register <-multiplicand

— A register<-

— Count <-n

• Product will be 2n bits in A Q registers Booth's Algorithm ControlLogic

• Bits of the multiplier are scanned one at aa time (the current bit Q0)

• As bit is examined the bit to the right is considered also (the previous bit Q-1)

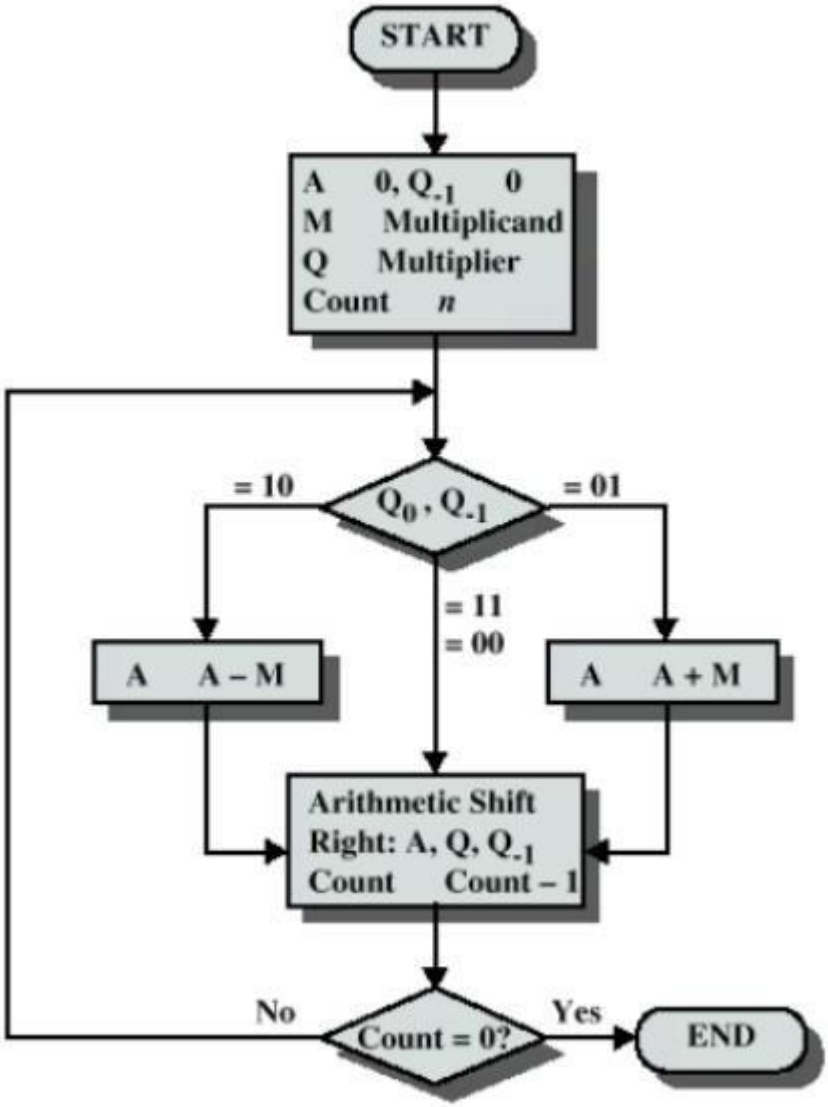


Fig. 2.6 Booth algrithm

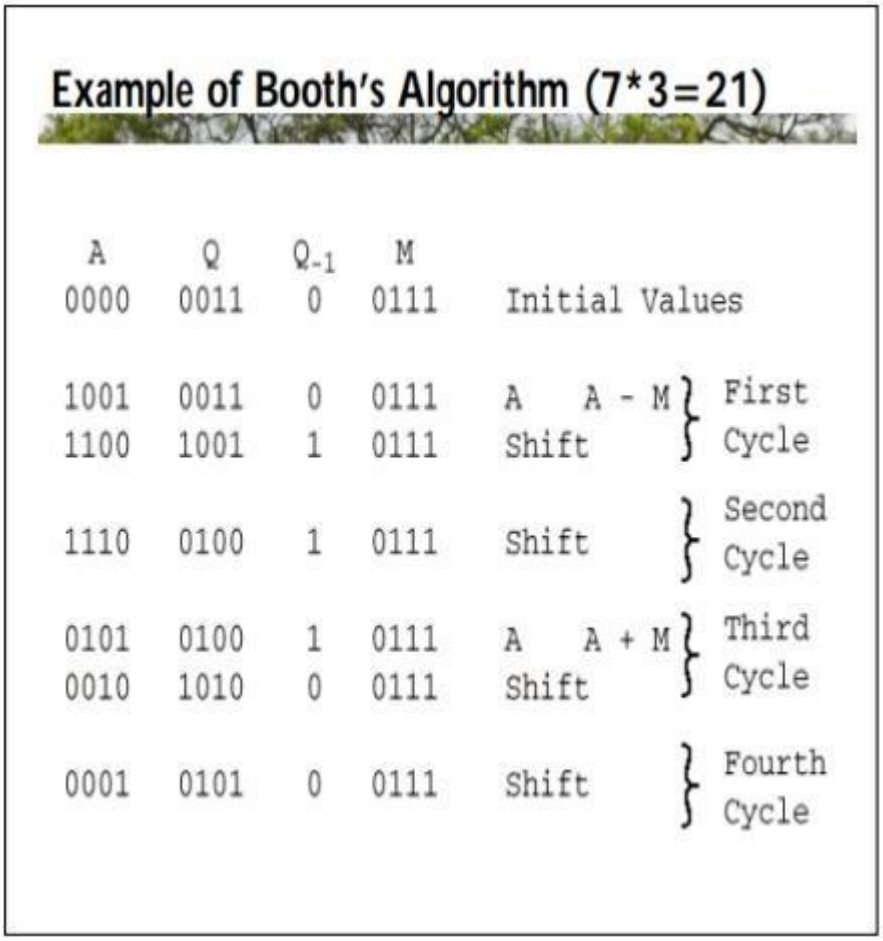
• Then:

00: Middle of a string of 0s, so no arithmetic operation.
01: End of a string of 1s, so add the multiplicand to the left half of the product (A).

10: Beginningofastringof1s,sosubtractthemultiplicandfromthelefthalfofthe product(A).
11: Middle of a string of 1s, so no arithmeticoperation.

• Then shift A, Q, bit Q-1 right one bit using an arithmeticshift

• In an arithmetic shift, the msbremains



DIVISION

The reciprocal operation of multiply is divide, an operation that is even less frequent and even morequirky.

$$
\begin{array}{rll}
 & 1001_{ten} & \text{Quotient} \\
\text{Divisor } 1000_{ten} & \overline{)\,1001010_{ten}} & \text{Dividend} \\
 & -1000 & \\
 & \overline{10} & \\
 & 101 & \\
 & 1010 & \\
 & -1000 & \\
 & \overline{10_{ten}} & \text{Remainder}
\end{array}
$$

It even offers the opportunity to perform a mathematically invalid operation: dividing by 0. The example is dividing 1,001,010 by 1000. The two operands (dividend and divisor) and the result(quotient)ofdivideareaccompaniedbyasecondresultcalledtheremainder.Hereisanother way to express the relationship between thecomponents:

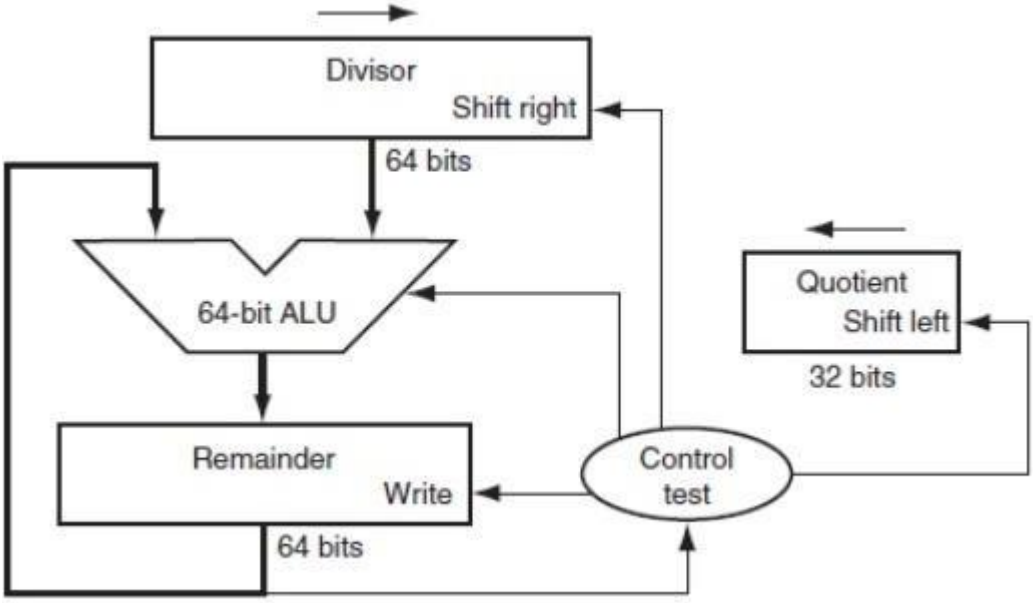


Fig. 2.8 First version of the Division hardware

Dividend = Quotient * Divisor + Remainder

where the remainder is smaller than the divisor. Infrequently, programs use the divide instruction just to get the remainder, ignoring the quotient. The basic grammar school division algorithm tries to see how big a number can be subtracted, creating a digit of the quotient oneach attempt. Binary numbers contain only 0 or 1, so binary division is restricted to these two choices, therebysimplifyingbinarydivision.Ifboththedividendanddivisorarepositiveandhencethe

quotient and the remainder are nonnegative. The division operands and both results are 32-bit values.

A Division Algorithm and Hardware

Initially,the32-bitQuotientregistersetto0.Eachiterationofthealgorithmneedstomove the divisor to the right one digit, start with the divisor placed in the left half of the 64-bit Divisor register and shift it right 1 bit each step to align it with the dividend. The Remainder register is initialized with the dividend. Figure shows three steps of the first divisionalgorithm.

Unlikeahuman,thecomputerisn’tsmartenoughtoknowinadvancewhetherthedivisoris smaller than the dividend. It must first subtract the divisor in step 1; If the result is positive, the divisor was smaller or equal to the dividend, so generate a 1 in the quotient (step 2a). If the result isnegative,thenextstepistorestoretheoriginalvaluebyaddingthedivisorbacktotheremainder and generate a 0 in the quotient (step 2b). The divisor is shifted right and then iterate again. The remainder and quotient will be found in their namesake registers after the iterations arecomplete.

Thefollowingfigureshowsthreestepsofthefirstdivisionalgorithm.Unlikeahuman,the computer isn’t smart enough to know in advance whether the divisor is smaller than the dividend.

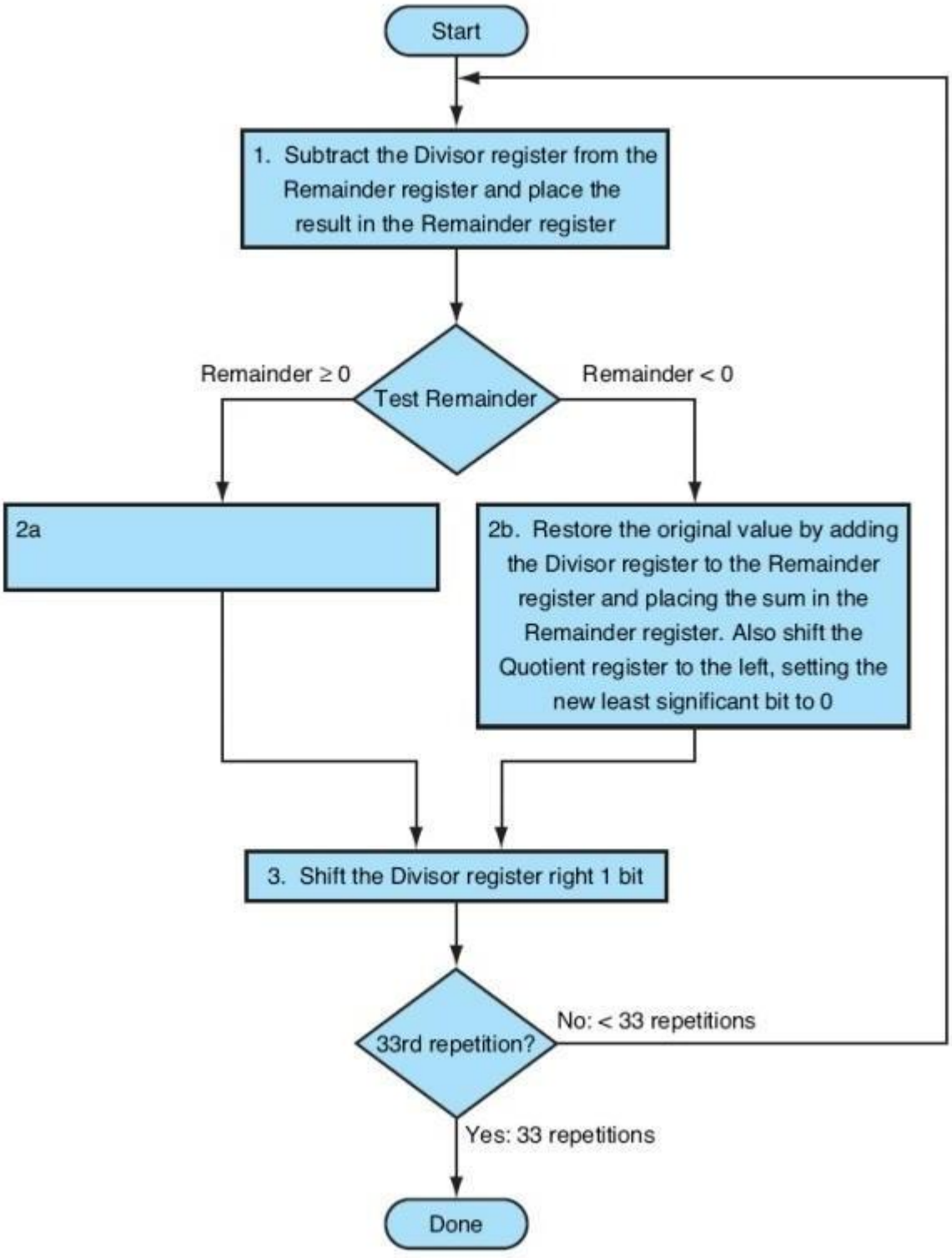


Fig. 2.9 Division Algorithm

It must first subtract the divisor in step 1; remember that this is how we performed the comparison in the set on less than instruction. If the result is positive, the divisor was smaller or equaltothedividend,sowegeneratea1inthequotient(step2a).Iftheresultisnegative,thenext stepistorestoretheoriginalvaluebyaddingthedivisorbacktotheremainderandgeneratea0in the quotient (step 2b). The divisor is shifted right and then we iterate again. The remainder and quotient will be found in their namesake registers after the iterations arecomplete.

Using a 4-bit version of the algorithm to save pages, let's try dividing 7_{10} by 2_{10}, or 0000 0111_2 by 0010_2.

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	1110 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	1111 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	1111 1111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	0000 0011
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	0000 0001
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

Fig. 2.10 Values of register in division algorithm

The above figure shows the value of each register for each of the steps, with the quotient being3tenandtheremainder1ten.Noticethatthetestinstep2ofwhethertheremainderispositive or negative simply tests whether the sign bit of the Remainder register is a 0 or 1. The surprising requirement of this algorithm is that it takes n + 1 steps to get the proper quotient andremainder.

This algorithm and hardware can be refined to be faster and cheaper. The speedup comes from shifting the operands and the quotient simultaneously with the subtraction. This refinement halvesthewidthoftheadderandregistersbynoticingwheretherareunusedportionsofregisters andadders.

SIGNED DIVISION

The one complication of signed division is that we must also set the sign of the remainder. Remember that the following equation must always hold:

Dividend = Quotient × Divisor + Remainder

To understand how to set the sign of the remainder, let's look at the example of dividing all the combinations of $\pm 7_{10}$ by $\pm 2_{10}$.

The first case is easy:

+7 ÷ +2: Quotient = +3, Remainder = +1 Checking the results:

7 = 3 × 2 + (+1) = 6 + 1

If we change the sign of the dividend, the quotient must change as well:

–7 ÷ +2: Quotient = –3

Rewriting our basic formula to calculate the remainder:

Remainder = (Dividend – Quotient × Divisor) = –7 – (–3 × +2) = –7–(–6) = –1

So,
–7 ÷ +2: Quotient = –3, Remainder = –1

Checking the results again:
–7 = –3 × 2 + (–1) = – 6 – 1

The following figure shows the revised hardware.

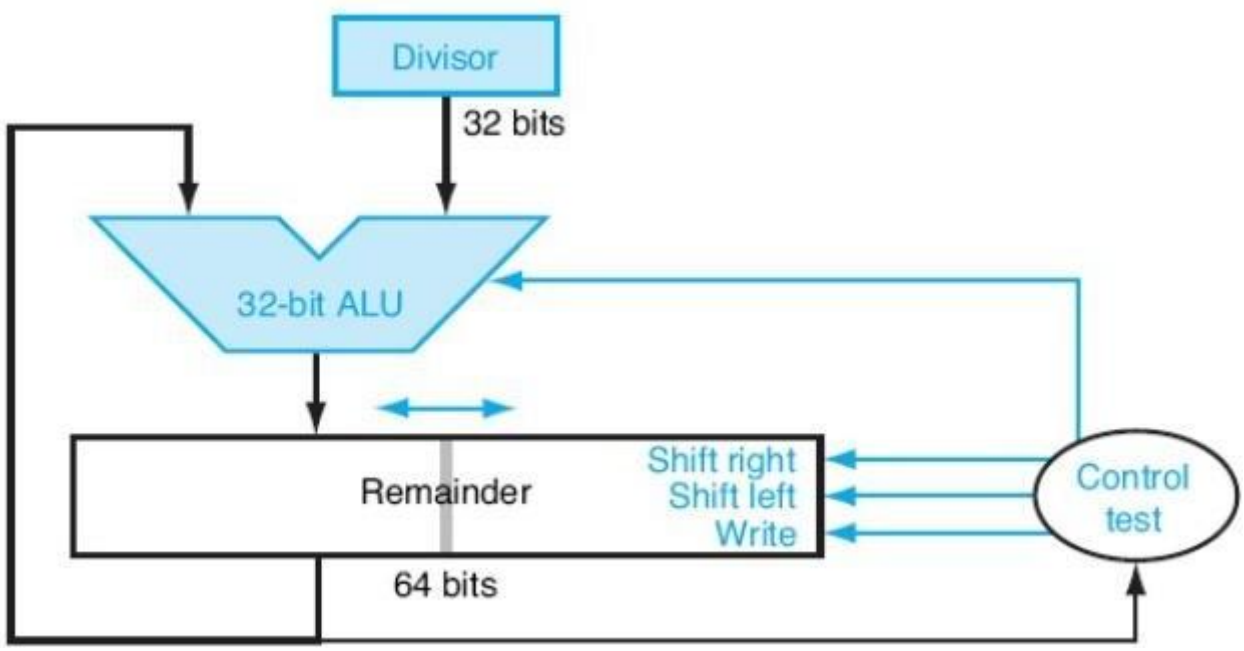


Fig. 2.11 Division hardware

The reason the answer isn't a quotient of –4 and a remainder of +1, which would also fit thisformula,isthattheabsolutevalueofthequotientwouldthenchangedependingonthesignof the dividend and the divisor! Clearly,if

$$-(x \div y) \neq (-x) \div y$$

programming would be an even greater challenge. This anomalous behavior is avoided by following the rule that the dividend and remainder must have the same signs, no matter what the signsofthedivisorandquotient.Wecalculatetheothercombinationsbyfollowingthesamerule:

$+7 \div -2$: Quotient = –3, Remainder = +1
$-7 \div -2$: Quotient = +3, Remainder = –1

Thus the correctly signed division algorithm negates the quotient if the signs of the operands are opposite and makes the sign of the nonzero remainder match the dividend.

Faster Division

Many adders can be used to speed up multiply, cannot be used to do the same trick for divide.Thereasonisthatitisneededtoknowthesignofthedifferencebeforeperformingthenext step of the algorithm, whereas with multiply we could calculate the 32 partial products immediately.

There are techniques to produce more than one bit of the quotient per step. The SRT division technique tries to guess several quotient bits per step, using a table lookup based on the upper bits of the dividend and remainder. It relies on subsequent steps to correct wrong guesses. A typical value today is 4 bits. The key is guessing the value to subtract. With binary division, there is only a single choice.

Thesealgorithmsuse6bitsfromtheremainderand4bitsfromthedivisor toindexatablethatdeterminestheguessforeachstep.Theaccuracyofthisfastmethoddependsonhavingproper values in the lookuptable.

Restoring and non restoring division algorithm

•Assume ─ X register k-bit dividend
• Assume ─ Y the k-bitdivisor
• Assume ─ S asign-bit

1. Start: Load 0 into accumulator k-bit A and dividend X is loaded into the k-bit quotient register MQ.

2. Step A : Shift 2 k-bit register pair A -MQleft
3. Step B: Subtract the divisor Y fromA.
4. Step C: If sign of A (msb) = 1, then reset MQ 0 (lsb) = 0 else set =1.
5. Steps D: If MQ 0 = 0 add Y (restore the effect of earliersubtraction).

6. Steps A to D repeat again till the total number of cyclic operations = k. At the end, A has the remainder and MQ hasthe

Step	S-flag *	First Register for A	Second Register for MQ	Action Taken	Number of operations (instructions)
Start	0	0b 0000	0b 0000	Clear S, A, MQ	3 for clearing C, A and M
	0	0b 0001	0b 1110	Load dividend X (lower k bits) between MQ_{k-1} and MQ_0 and dividend higher bits in A	2 for loading A and MQ
Step 0A	0	0011	1100	Shift left S-A-M	2
Step 0B	0	0000	1100	Subtract Y from S-A, result in S-A	1
Step 0C	0	0000	1101	$MQ_0 = 1$ as S = 0	1
Step 0D	0	0000	1101	Skip restore by adding as S = 0	1 (test S)
Step 1A	0	0001	1010	Shift left S-A-M	2
Step 1B	1	1110	1010	Subtract Y from S-A, result in S-A	1
Step 1C	1	1110	1010	$MQ_0 = 0$ as S = 1	1
Step 1D	0	0001	1010	Add Y into S-A to restore as S = 1	1
Step 2A	0	0011	0100	Shift left S-A-M	2
Step 2B	0	0000	0100	Subtract Y from S-A, result in S-A	1
Step 2C	0	0000	0101	$MQ_0 = 1$ as S = 0	1
Step 2D	0	0000	0101	Skip restore as S = 0	1(test S)
Step 3A	0	0000	1010	Shift left S-A-M	2
Step 3B	1	1101	1010	Subtract Y from S-A, result in S-A	1
Step 3C	1	1101	1010	$MQ_0 = 0$ as S = 1	1
Step 3D	0	0000	1010	Add Y into S-A to restore as S = 1	1
Answer	0	Remainder = 0,		Quotient Decimal 10	Total 25

* after the left shift from *msb* of A.

Fig. 2.12 Division of 4-bit number by 7-bit dividend

Division using Non-restoring Algorithm

• Assume ─ that there is an accumulator and MQ register, each of k-bits • MQ 0, (lsb of MQ) bit gives the quotient, which is saved after a subtraction oraddition

• Total number of additions or subtractions are k-only and total number of shifts = k plus one addition for restoring remainder ifneeded

• Assume ─ that X register has (2 k−1) bit for dividend and Y has the k-bitdivisor

• Assume ─ a sign-bit S shows thesign

1. Load (upper half k −1 bits of the dividend X) into accumulator k-bit A and loaddividend X (lower half bits into the lower k bits at quotient registerMQ

• Reset sign S =0

• Subtract the k bits divisor Y from S-A (1 plus k bits) and assign MQ 0 as perS

2. If sign of A, S = 0, shift S plus 2 k-bit register pair A-MQ left and subtract the k bits divisor Y from S-A (1 plus k bits); else if sign of A, S = 1, shift S plus 2 k-bit registerpair A - MQ left and add the divisor Y into S-A (1 plus kbits)

• Assign MQ 0 asper

Step	S-flag *	First Register for *A*	Second Register for MQ	Action Taken	Number operation (instructi
Start	0	0b0000	0b0000	Clear S, A, MQ	3 for clearing A and
	0	0b0001	0b1110	Load dividend X (lower *k* bits) in MQ_{k-1} and MQ_0 and dividend higher k–1 bits in *A*	2 for loa A and M
Step 0A	1	1110	1110	Subtract *Y* from S-*A*, because S = 0 result in S-*A*	1
Step 0B	1	1110	1110	MQ_0 = 0 as S = 1	1
Step 0C	1	1101	1100	Shift left S-A-M	2
Step 1A	0	0000	1100	Add Y into S-A, because S = 1	1
Step 1B	0	0000	1101	MQ_0 = 1 as S = 0	1
Step 1C	0	0001	1010	Shift left S-A-M	2
Step 2A	1	1110	1010	Subtract Y into S-A, because S = 0	1
Step 2B	1	1110	1010	MQ_0 = 0 as S = 1	1
Step 2C	1	1101	0100	Shift left S-A-M	2
Step 3A	1	0000	0100	Add Y into S-A, because S = 1	1
Step 3B	0	0000	0101	MQ_0 = 1 as S = 0	1
Step 3C	0	0000	1010	Shift C-A-M	2
Last	0	0000	1010	Do not Add Y into S-A, because S = 0 and make no change in MQ_0	1
Answer	0	Remainder = 0,		Quotient Decimal 10	Total 22

Fig. 2.13 Division using Non-restoring Algorithm

3. Repeat step 2 again till the total number of operations =k.

4. IfatthelaststeP,thesignofAinS=1,thenaddYintoS-AtoleavethecorrectremainDer into A and also assign MQ 0 as per S, else donothing.

5. A has the remainder and MQ has thequotient

FLOATING POINT OPERATIONS

The scientific notation has a single digit to the left of the decimal point. A number in scientific notation that has no leading 0s is called a normalized number, which is the usualway to write it. Foating point - Computer arithmetic that represents numbers in which the binary point is not fixed. Floating-point numbers are usually a multiple of the size of aword.

TherepresentationofaMIPSfloating-pointnumberisshownbelow,wheresisthesignof the floating-point number (1 meaning negative), exponent is the value of the 8-bit exponent field (includingthesignoftheexponent),andfractionisthe23-bitnumber.Thisrepresentationiscalled sign and magnitude, since the sign has a separate bit from the rest of thenumber.

A standard scientific notation for reals in normalized form offers three advantages. • It simplifies exchange of data that includes floating-point numbers; It simplifies the floating-point arithmetic algorithms to know that numbers will always be in thisform;

- Itincreasestheaccuracyofthenumbersthatcanbestoredinaword,sincetheunnecessary leading 0s are replaced by real digits to the right of the binarypoint.

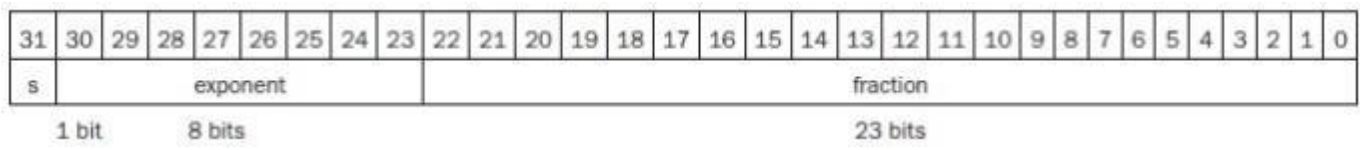


Fig. 2.14 Scientific notation

Floating point addition

Step 1. To be able to add these numbers properly, align the decimal point of the number that has thesmallerexponent.Hence,weneedaformofthesmallernumber,1.610tenx10–1,thatmatches the larger exponent. We obtain this by observing that there are multiple representations of an unnormalized floating-point number in scientificnotation:

Step 1.1.610ten $x10^{-1}$ = 0.1610ten x100 = 0.01610ten $x10^{1}$

Step 2. Next comes the addition of the significands: 9.999ten+ 0.016ten The sum is 10.015ten $x10^{1}$.

Step 3. This sum is not in normalized scientific notation, so we need to adjust it: 10.015tenx10^{1} = 1.0015ten $x10^{2}$

Thus, after the addition we may have to shift the sum to put it into normalized form, adjustingtheexponentappropriately.

SUB WORD PARALLELISM

A subword is a lower precision unit of data contained within a word. In subword parallelism, multiple subwords are packed into a word and then process whole words. With the appropriate subword boundaries this technique results in parallel processing of subwords. Since the same instruction is applied to all subwords within the word, This is a form of SIMD(Single Instruction Multiple Data) processing.

It is possible to apply subword parallelism to noncontiguous subwords of different sizes within a word. In practical implementation is simple if subwords are same size and they are contiguous within a word. The data parallel programs that benefit from subword parallelism tend to process data that are of the same size.

For example if word size is 64bits and subwords sizes are 8,16 and 32 bits. Hence an instruction operates on eight 8bit subwords, four 16bit subwords, two 32bit subwords or one 64bit subword in parallel.

- Subword parallelism is an efficient and flexible solution for media processing because algorithm exhibit a great deal of data parallelism on lower precisiondata.
- It is also useful for computations unrelated to multimedia that exhibit data parallelism on lower precisiondata.
- Graphics and audio applications can take advantage of performing simultaneous operations on shortvectors

UNIT III THE PROCESSOR AND CONTROL UNIT

BASIC MIPS IMPLEMENTATION

1. Instruction fetch cycle(IF):

IR = Mem[PC];

NPC = PC + 4; Operation:

Send out the PC and fetch the instruction from memory into the instruction register (IR).
Increment the PC by 4 to address the next sequential instruction.

IR - holds instruction that will be needed on subsequent clock cycles
Register NPC - holds next sequential PC.

2. Instruction decode/register fetch cycle(ID):

A = Regs[rs];

B = Regs[rt];

Imm = sign-extended immediate field of IR; Operation:

Decode instruction and access register file to read the registers (rs and rt -register specifiers).
Outputs of general purpose registers are read into 2 temporary registers (A and B) for use in later clock cycles.

Lower 16 bits of IR are sign extended and stored into the temporary register Imm, for use in the next cycle.

3. Execution/effective address cycle(EX):

* ALU operates on the operands prepared in the prior cycle, performing one of four functions depending on the MIPS instructiontype.

i) Memory reference:
ALUOutput = A +Imm;
ii) Register-Register ALUinstruction:

ALUOutput = A func B;
Operation:

a) ALU performs the operation specified by the function code on the value in registerA and in register B.
b) Result is placed in temporary registerALUOutput
c) iii) Register-Immediate ALUinstruction:

ALUOutput = A op Imm;
Operation:

a) ALUperformsoperationspecifiedbytheopcodeonthevalueinregisterAandregister Imm.

b) Result is placed in temporary registerALUOutput.

iv)Branch:

ALUOutput = NPC + (Imm<<
2); Cond = (A == 0)
Operation:

a) ALU adds NPC to sign-extended immediate value in Imm, which is shifted left by 2 bits to create a word offset, to compute address of branchtarget.

b) Register A, which has been read in the prior cycle, is checked to determine whether branch istaken.

c) Considering only one form of branch (BEQZ), the comparison is against0.

4. Memory access/branch completion cycle (MEM):

* PC is updated for all instructions: PC = NPC; i. Memoryreference:

LMD = Mem[ALUOutput] or
Mem[ALUOutput] = B;

Operation:

a) Access memory ifneeded.

b) Instruction is load-data returns from memory and is placed in LMD (load memory data)

c) Instruction is store-data from the B register is written intomemory

ii.Branch:

if (cond) PC = ALUOutput

Operation: If the instruction branches, PC is replaced with the branch destination address in register ALUOutput.

5.Write-back cycle(WB):

* Register-Register ALU instruction: Regs[rd] =ALUOutput;
* Register-Immediate ALU instruction: Regs[rt] =ALUOutput;
* Loadinstruction:

Regs[rt] = LMD;

Operation: Write the result into register file, depending on the effective opcode.

BUILDING DATA PATH AND CONTROL IMPLEMENTATION SCHEME

Datapath

- Components of the processor that perform arithmetic operations and holdsdata.

Control

· Componentsoftheprocessorthatcommandsthedatapath,memory,I/Odevicesaccording to the instructions of thememory.

Building a Datapath

· Elements that process data and addresses in the CPU - Memories, registers,ALUs.
· MIPS datapath can be built incrementally by considering only a subset ofinstructions
· 3 main elements are

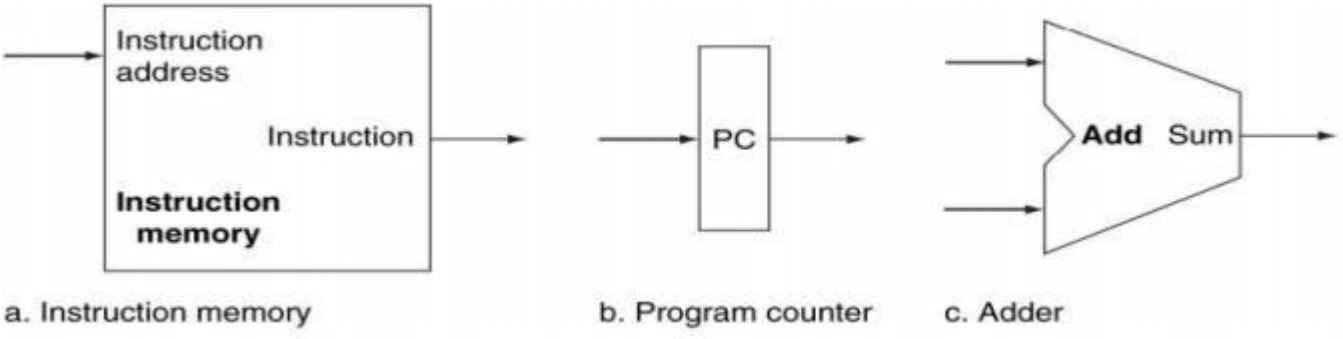


Fig. 3.1 Datapath

· Amemoryunittostoreinstructionsofaprogramandsupplyinstructionsgivenanaddress. Needs to provide only read access (once the program is loaded).- No control signal is needed

· PC (Program Counter or Instruction address register) is a register that holds the addressof the currentinstruction

  Ø A new value is written to it every clock cycle. No control signal is required to enable write
  Ø Adder to increment the PC to the address of the next instruction

· An ALU permanently wired to do only addition. No extra control signalrequired

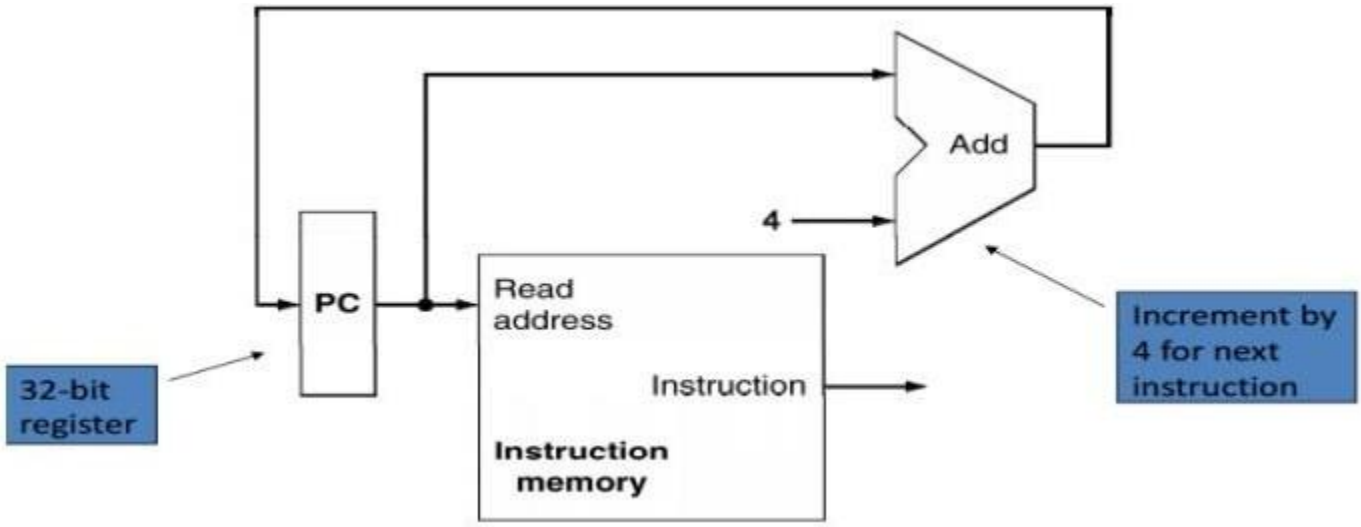


Fig. 3.2 Datapath portion for Instruction Fetch

Types of Elements in the Datapath

State element:

- A memory element, i.e., it contains astate
- E.g., program counter, instruction memory Combinationalelement:
- Elements that operate onvalues
- Eg adder ALU E.g. adder,ALU

Elements required by the different classes of instructions

- Arithmetic and logicalinstructions
- Data transferinstructions
- Branch instructions

R-Format ALUInstructions

- E.g., add $t1, $t2,$t3
- Perform arithmetic/logicaloperation
- Read two register operands and write register result

Register file:

- A collection of theregisters
- Any register can be read or written by specifying the number of theregister
- Contains the register state of thecomputer

Read from register

- 2 inputs to the register file specifying thenumbers
  - 5 bit wide inputs for the 32registers

- 2 outputs from the register file with the readvalues
  - 32 bit wide

- For all instructions. No control required.

Write to register file

- 1 input to the register file specifying the number 5 bit wide inputs for the 32registers
- 1 input to the register file with the value to be written 32 bitwide

- Only for some instructions. RegWritecontrolsignal.

ALU

- Takes two 32 bit input and produces a 32 bitoutput
- Also, sets one-bit signal if the results is0

- The operation done by ALU is controlled by a 4 bit control signal input. This is set according to the instruction.

PIPELINING
Basicconcepts

- Pipeliningisanimplementationtechniquewherebymultipleinstructionsareoverlappedin execution

- Takes advantage ofparallelism
- Key implementation technique used to make fastCPUs.
- A pipeline is like an assemblyline.

Inacomputerpipeline,eachstepinthepipelinecompletesapartofaninstruction.Likethe assembly line, different steps are completing different parts of different instructions in parallel. Each of these steps is called a pipe stage or a pipesegment.

- The stages are connected one to the next to form a pipe—instructions enter at one end, progress through the stages, and exit at the otherend.

- The throughput of an instruction pipeline is determined by how often an instruction exits thepipeline.

- Thetimerequiredbetweenmovinganinstructiononestepdownthepipelineisaprocessor cycle..

- If the starting point is a processor that takes 1 (long) clock cycle per instruction, then pipelining decreases the clock cycletime.

- Pipeline for an integer subset of a RISC architecture that consists of load-store word, branch, and integer ALUoperations.

Every instruction in this RISC subset can be implemented in at most 5 clock cycles. The 5 clock cycles are as follows.

1. Instruction fetch cycle(IF):

Send the program counter (PC) to memory and fetch the current instruction from memory. PC=PC+4

2. Instruction decode/register fetch cycle(ID):

- Ø Decode the instruction and read theregisters.
- Ø Do the equality test on the registers as they are read, for a possiblebranch.

- Ø Compute the possible branch target address by adding the sign-extended offset to the incrementedPC.

- Ø Decodingisdoneinparallelwithreadingregisters,whichispossiblebecausetheregister specifiers are at a fixed location in a RISC architecture,known as fixed-fielddecoding

3. Execution/effective addresscycle(EX):

The ALU operates on the operands prepared in the prior cycle, performing one of three functions depending on the instruction type.

- Ø Memory reference: The ALU adds the base register and the offset to form the effective address.

- Ø Register-Register ALU instruction: The ALU performs the operation specified by the ALU opcode on the values read from the registerfile
- Ø Register-Immediate ALU instruction: The ALU performs the operation specified by the ALUopcodeonthefirstvaluereadfromtheregisterfileandthesign-extendedimmediate.

4. Memoryaccess(MEM):

- Ø If the instruction is a load, memory does a read using the effective address computed in the previouscycle.

- Ø If it is a store, then the memory writes the data from the second register read from the register file using the effectiveaddress.

5. Write-back cycle(WB):
Register-Register ALU instruction or Load instruction: Write the result into the register file, whether it comes from the memory system (for a load) or from the ALU (for an ALU instruction).

PIPELINED DATA PATH AND CONTROL
The Classic Five-Stage Pipeline for a RISC Processor

Each of the clock cycles from the previous section becomes a pipe stage—a cycle in the pipeline.

Each instruction takes 5 clock cycles to complete, during each clock cycle the hardware will initiate a new instruction and will be executing some part of the five different instructions.

3 observations:

1. Use separate instruction and data memories, which implement with separate instruction and datacaches.

2. The register file is used in the two stages: one for reading in ID and one for writing in WB, need to perform 2 reads and one write every clockcycle.

3. Does not deal with PC, To start a new instruction every clock, we must increment and store the PC every clock, and this must be done during the IF stage in preparation for the next instruction.

To ensure that instructions in different stages of the pipeline do not interfere with one another. This separation is done by introducing pipeline registers between successive stages of the

pipeline, so that at the end of a clock cycle all the results from a given stage are stored into a register that is used as the input to the next stage on the next clock cycle.

HANDLING DATA HAZARDS & CONTROL HAZARDS

Hazards:Preventthenextinstructionintheinstructionstreamfromexecutingduringitsdesignatedclock cycle.
* Hazards reduce the performance from the ideal speedup gained bypipelining.
3 classes ofhazards:

- Ø Structural hazards: arise from resource conflicts when the hardware cannot support all possible combinations of instructions simultaneously in overlappedexecution.

- Ø Data hazards: arise when an instruction depends on the results of a previous instruction in a way that is exposed by the overlapping of instructions in thepipeline.

- Ø Control hazards: arise from the pipelining of branches and other instructions that change the PC.

Performance of Pipelines with Stalls
* A stall causes the pipeline performance to degrade from the ideal performance.

Speedup from pipelining = [1/ (1+ pipeline stall cycles per instruction)] *Pipeline

$$\text{Speedup from pipelining} = \frac{1}{1 + \text{pipeline stall cycles per instruction}} * \text{Pipeline depth}$$

Structural Hazards

* When a processor is pipelined, the overlapped execution of instructions requires pipelining of functional units and duplication of resources to allow all possible combinations of instructions in thepipeline.

* If some combination of instructions cannot be accommodated because of resource conflicts, the processor is said to have a structuralhazard.
* Instances:

§ When functional unit is not fully pipelined, Then a sequence of instructions using that unpipelined unit cannot proceed at the rate of one per clock cycle.

§ when some resource has not been duplicated enough to allow all combinations of instructions in the pipeline to execute.

* To Resolve thishazard,

§ Stall the the pipeline for 1 clock cycle when the data memory access occurs. A stall is commonly called a pipeline bubble or just bubble, since it floats through the pipeline taking space but carrying no usefulwork.

Data Hazards

* A major effect of pipelining is to change the relative timing of instructions by overlapping their execution. This overlap introduces data and controlhazards.

* Data hazards occur when the pipeline changes the order of read/write accesses to operandssothattheorderdiffersfromtheorderseenbysequentiallyexecutinginstructions on an unpipelinedprocessor.

Minimizing Data Hazard Stalls by Forwarding

* The problem solved with a simple hardware technique called forwarding (also called bypassing and sometimes short-circuiting). Forwards worksas:

§ The ALU result from both the EX/MEM and MEM/WB pipeline registers is always fed back to the ALU inputs.

§If the forwarding hardware detects that the previous ALU operation has written the register corresponding to a source for the current ALU operation, control logic selects the forwarded result as the ALU input rather than the value read from the register file.

Data Hazards Requiring Stalls

* The load instruction has a delay or latency that cannot be eliminated by forwarding alone.Instead,weneedtoaddhardware,calledapipelineinterlock,topreservethecorrect executionpattern.

* A pipeline interlock detects a hazard and stalls the pipeline until the hazard iscleared.

* This pipeline interlock introduces a stall or bubble. The CPI for the stalled instruction increases by the length of thestall.

Branch Hazards

* Control hazards can cause a greater performance loss for our MIPS pipeline . When a branch is executed, it may or may not change the PC to something other than its current value plus4.

* IfabranchchangesthePCtoitstargetaddress,itisatakenbranch;ifitfallsthrough,it is not taken, oruntaken.

Reducing Pipeline Branch Penalties

* Simplestschemetohandlebranchesistofreezeorflushthepipeline,holdingordeleting any instructions after the branch until the branch destination isknown.

* A higher-performance, and only slightly more complex, scheme is to treat everybranch as not taken, simply allowing the hardware to continue as if the branch werenot

executed.Thecomplexityofthisschemearisesfromhavingtoknowwhenthestatemight be changed by an instruction and how to “back out” such achange.

* In simple five-stage pipeline, this predicted-not-taken or predicted untaken scheme is implementedbycontinuingtofetchinstructionsasifthebranchwereanormalinstruction.

§ The pipeline looks as if nothing out of the ordinary is happening.

§ If the branch is taken, however, we need to turn the fetched instruction into a no-op and restart the fetch at the targetaddress.

* Analternativeschemeistotreateverybranchastaken.Assoonasthebranchisdecoded and the target address is computed, we assume the branch to be taken and begin fetching and executing at the target**Performance of BranchSchemes**

Pipeline speedup = Pipeline depth / [1+ Branch frequency × Branch penalty]

$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Branch frequency} \times \text{Branch penalty}}$$

The branch frequency and branch penalty can have a component from both unconditional and conditional branches.

EXCEPTIONS

Controlisthemostchallengingaspectofprocessordesign:itisboththehardestparttoget right and the hardest part to make fast. One of the hardest parts of control isimplementing exceptions and interrupts—events other than branches or jumps that change the normal flow of instruction execution. An exception is an unexpected event from within the processor;arithmeticoverflowisanexampleofanexception.Aninterruptisaneventthat alsocausesanunexpectedchangeincontrolflowbutcomesfromoutsideoftheprocessor. Interrupts are used by I/O devices to communicate with theprocessor.

Manyarchitecturesandauthorsdonotdistinguishbetweeninterruptsandexceptions,often using the older name interrupt to refer to both types of events. MIPS convention uses the term exception to refer to any unexpected change in control flow without distinguishing whether the cause is internal or external; we use the term interrupt only when the event is externallycaused.

The Intel IA-32 architecture uses the word interrupt for all these events. Interrupts were initially created to handle unexpected events like arithmetic overflow and to signal requests for servicefromI/Odevices.Thesamebasicmechanismwasextendedtohandleinternallygenerated exceptions as well. Here are some examples showing whether the situation is generatedinternally by the processor or externallygenerated:

Type of event

- I/O device request -External
- Invoke the operating system from user program -Internal
- Arithmetic overflow -Internal
- Using an undefined instruction -Internal
- Hardware malfunctions -Either

The operating system knows the reason for the exception by the address at which it is initiated. The addresses are separated by 32 bytes or 8 instructions, and the operating systemmustrecordthereasonfortheexceptionandmayperformsomelimitedprocessing inthissequence.Whentheexceptionisnotvectored,asingleentrypointforallexceptions can be used, and the operating system decodes the status register to find thecause.

Fortheoperatingsystemtohandletheexception,itmustknowthereasonfortheexception, in addition to the instruction that caused it. There are two main methods used to communicatethe reason for an exception. The method usedin theMIPSarchitectureistoincludeastatusregister(calledtheCauseregister),whichholdsafield thatindicatesthereasonfortheexception.Asecondmethodistousevectoredinterrupts.Ina vectored interrupt, the address to which control is transferred is determined by the cause of the exception.

UNIT IV PARALLELISM

INSTRUCTION-LEVEL-PARALLELISM

Allprocessorssinceabout1985usepipeliningtooverlaptheexecutionofinstructionsand improve performance. This potential overlap among instructions is called instruction-level parallelism (ILP), since the instructions can be evaluated inparallel.

There are two largely separable approaches to exploiting ILP: an approach that relies on hardware to help discover and exploit the parallelism dynamically, and an approach that relies on software technology to find parallelism, statically at compile time. Processors using the dynamic, hardware-based approach, including the Intel Pentium series, dominate in the market; those using thestaticapproach,includingtheIntelItanium,havemorelimitedusesinscientificorapplication-specificenvironments.

The value of the CPI (cycles per instruction) for a pipelined processor is the sum of the base CPI and all contributions from stalls: Pipeline CPI = Ideal pipeline CPI + Structural stalls + Data hazard stalls + Control stalls

The ideal pipeline CPI is a measure of the maximum performance attainable by the implementation. By reducing each of the terms of the right-hand side to minimize the overall pipeline CPI or, alternatively, increase the IPC (instructions per clock).

The simplest and most common way to increase the ILP is to exploit parallel- ism among iterations of a loop. This type of parallelism is often called loop-level parallelism.There are a numberoftechniquesforconvertingsuchloop-levelparallelismintoinstruction-levelparallelism.
Basically, such techniques work by unrolling the loop either statically by the compiler or dynamically by the hardware. An important alternative method for exploiting loop-level parallelism is the use of vector instructions . A vector instruction exploits data- level parallelism by operating on data items inparallel.

PARALLEL PROCESSING CHALLENGES

Limitations of ILP

The Hardware Model

AnidealprocessorisonewhereallconstraintsonILPareremoved.TheonlylimitsonILP insuchaprocessorarethoseimposedbytheactualdataflowsthrougheitherregistersormemory.

The assumptions made for an ideal or perfect processor are as follows:

1.Register renaming

—Thereareaninfinitenumberofvirtualregistersavailable,andhenceallWAWandWAR hazardsareavoidedandanunboundednumberofinstructionscanbeginexecutionsimultaneously.

2. Branchprediction

—Branch prediction is perfect. All conditional branches are predicted exactly.

3. Jump prediction

—All jumps (including jump register used for return and computed jumps) are perfectly predicted. When combined with perfect branch prediction, this is equivalent to having a processorwithperfectspeculationandanunboundedbufferofinstructionsavailableforexecution.

4. Memory address aliasanalysis

—All memory addresses are known exactly, and a load can be moved before a store provided that the addresses are not identical. Note that this implements perfect address alias analysis.

5. Perfectcaches

—Allmemoryaccessestake1clockcycle.Inpractice,superscalarprocessorswilltypically consume large amounts of ILP hiding cache misses, making these results highlyoptimistic.

To measure the available parallelism, a set of programs was compiled and optimized with the standard MIPS optimizing compilers. The programs were instrumented and executed to produce a trace of the instruction and data references. Every instruction in the trace is then scheduledasearlyaspossible,limitedonlybythedatadependences.Sinceatraceisused,perfect branch prediction and perfect alias analysis are easy to do. With these mechanisms, instructions may bescheduled much earlier than they would otherwise, moving across large numbers of instructionsonwhichtheyarenotdatadependent,includingbranches,sincebranchesareperfectly predicted.

The effects of various assumptions are given before looking at some ambitious but realizable processors.

Limitations on the Window Size and Maximum Issue Count

To build a processor that even comes close to perfect branch prediction and perfect alias analysisrequiresextensivedynamicanalysis,sincestaticcompiletimeschemescannotbeperfect. Of course, most realistic dynamic schemes will not be perfect, but the use of dynamic schemes will provide the ability to uncover parallelism that cannot be analyzed by static compile time analysis.Thus,adynamicprocessormightbeabletomorecloselymatchtheamountofparallelism uncovered by our idealprocessor.

The Effects of Realistic Branch and Jump Prediction

Ouridealprocessorassumesthatbranchescanbeperfectlypredicted:Theoutcomeof any branch in the program is known before the first instruction is executed! Of course, no real processor can ever achievethis.

We assume a separate predictor is used for jumps. Jump predictors are important primarily with the most accurate branch predictors, since the branch frequency is higher and the accuracy of the branch predictors dominates.

1. Perfect —All branches and jumps are perfectly predicted at the start of execution.
2. Tournament-based branch predictor —The prediction scheme uses a correlating 2-bit

predictor and a noncorrelating 2-bit predictor together with a selector, which chooses the best predictor for eachbranch.

The Effects of Finite Registers

Our ideal processor eliminates all name dependences among register references using an infinitesetofvirtualregisters.Todate,theIBMPower5hasprovidedthelargestnumbersofvirtual registers: 88 additional floating-point and 88 additional integer registers, in addition to the 64 registers available in the base architecture. All 240 registers are shared by two threads when executing in multithreading mode, and all are available to a single thread when in single-thread mode.

The Effects of Imperfect Alias Analysis

Our optimal model assumes that it can perfectly analyze all memory dependences, as well as eliminate all register name dependences. Of course, perfect alias analysis is not possible in practice: The analysis cannot be perfect at compile time, and it requires a potentially unbounded number of comparisons at run time (since the number of simultaneous memory references is unconstrained).

The three models are

1. Global/stack perfect—This model does perfect predictions for global and stack referencesandassumesallheapreferencesconflict.Thismodelrepresentsanidealizedversionof thebestcompiler-basedanalysisschemescurrentlyinproduction.Recentandongoingresearchon alias analysis for pointers should improve the handling of pointers to the heap in thefuture.

2. Inspection—This model examines the accesses to see if they can be determined not to interfere at compile time. For example, if an access uses R10 as a base register with an offset of 20, then another access that uses R10 as a base register with an offset of 100 cannot interfere, assuming R10 could not have changed. In addition, addresses based on registers that point to different allocation areas (such as the global area and the stack area) are assumed never to alias. This analysis is similar to that performed by many existing commercial compilers, though newer compilers can do better, at least for looporientedprograms.

3. None—All memory references are assumed to conflict. As you might expect, for the FORTRANprograms(wherenoheapreferencesexist),thereisnodifferencebetweenperfectand global/stack perfectanalysis

FLYNN'S CLASSIFICATION

In 1966, Michael Flynn proposed a classification for computer architectures based on the number of instruction steams and data streams (Flynn’sTaxonomy).

- Flynn uses the stream concept for describing a machine'sstructure.

- A stream simply means a sequence of items (data orinstructions).
- The classification of computer architectures based on the number of instruction steams and data streams (Flynn'sTaxonomy).

Flynn's Taxonomy

- SISD: Single instruction singledata
  - Classical von Neumannarchitecture
- SIMD: Single instruction multipledata
- MISD: Multiple instructions singledata
  - Non existent, just listed forcompleteness
- MIMD: Multiple instructions multipledata
  - Most common and general parallelmachine

SISD

- SISD (Singe-Instruction stream, Singe-Datastream)

- SISDcorrespondstothetraditionalmono-processor(vonNeumanncomputer).Asingle data stream is being processed by one instructionstream

A single-processor computer (uni-processor) in which a single stream of instructions is generated from theprogram.

SIMD

- SIMD (Single-Instruction stream, Multiple-Datastreams)
- Eachinstructionisexecutedonadifferentsetofdatabydifferentprocessorsi.emultiple processing units of the same type process on multiple-datastreams.
- This group is dedicated to array processingmachines.
- Sometimes, vector processors can also be seen as a part of thisgroup.

MISD

- MISD (Multiple-Instruction streams, Singe-Datastream)
- Each processor executes a different sequence ofinstructions.
- In case of MISD computers, multiple processing units operate on one single-datastream
- In practice, this kind of organization has never beenused

MIMD

- MIMD (Multiple-Instruction streams, Multiple-Data streams)
- Each processor has a separateprogram.
- An instruction stream is generated from eachprogram.
- Each instruction operates on differentdata.

This last machine type builds the group for the traditional multi-processors. Several processing units operate on multiple-datastreams

HARDWARE

Exploiting Thread-Level Parallelism within a Processor.

- Multithreadingallowsmultiplethreadstosharethefunctionalunitsofasingleprocessor in an overlappingfashion.

- Topermitthissharing,theprocessormustduplicatetheindependentstateofeachthread.

- Forexample,aseparatecopyoftheregisterfile,aseparatePC,andaseparatepagetable are required for eachthread.

There are two main approaches to multithreading.

1. Fine-grained multithreading switches between threads on each instruction, causing the executionofmultiplesthreadstobeinterleaved.Thisinterleavingisoftendoneinaround- robin fashion, skipping any threads that are stalled at thattime.

2. Coarse-grained multithreading was invented as an alternative to fine-grained multithreading. Coarse-grained multithreading switches threads only on costly stalls,such as level two cache misses. This change relieves the need to have thread-switching be essentiallyfreeandismuchlesslikelytoslowtheprocessordown,sinceinstructionsfrom other threads will only be issued, when a thread encounters a costlystall.

Simultaneous Multithreading:

- Converting Thread-Level Parallelism into Instruction-LevelParallelism.

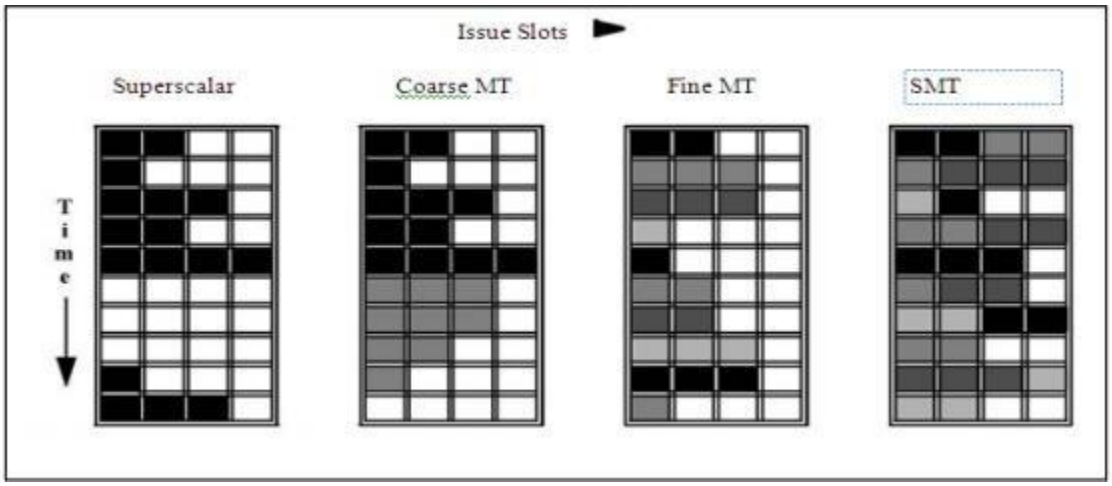
- Simultaneous multithreading (SMT) is a variation on multithreading that uses the resources of a multiple issue, dynamically-scheduled processor to exploit TLP at thesame time it exploitsILP.

- ThekeyinsightthatmotivatesSMTisthatmodernmultiple-issueprocessorsoftenhave more functional unit parallelism available than a single thread can effectivelyuse.

- Furthermore,withregisterrenaminganddynamicscheduling,multipleinstructionsfrom independent threads can be issued without regard to the dependences among them; the resolution of the dependences can be handled by the dynamic schedulingcapability.

The following figure illustrates the differences in a processor's ability to exploit the resources of a superscalar for the following processor configurations:

- a superscalar with no multithreadingsupport,
- a superscalar with coarse-grainedmultithreading,
- a superscalar with fine-grained multithreading,and
- a superscalar with simultaneousmultithreading.



- In the superscalar without multithreading support, the use of issue slots is limited by a lack of ILP.
- Inthecoarse-grainedmultithreadedsuperscalar,thelongstallsarepartiallyhiddenby switching to another thread that uses the resources of the processor.

- Inthefine-grainedcase,theinterleavingofthreadseliminatesfullyemptyslots.Because only one thread issues instructions in a given clock cycle.In the SMT case, thread-level parallelism (TLP) and instruction-level parallelism (ILP) are exploited simultaneously; with multiple threads using the issue slots in a single clockcycle.

- Theabovefiguregreatlysimplifiestherealoperationoftheseprocessorsitdoesillustrate the potential performance advantages of multithreading in general and SMT inparticular.

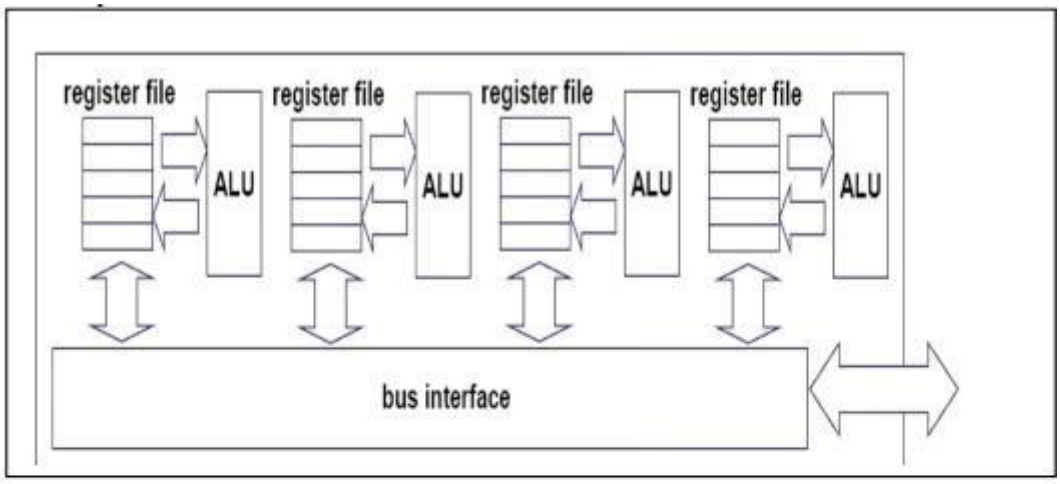
Multi-core processor

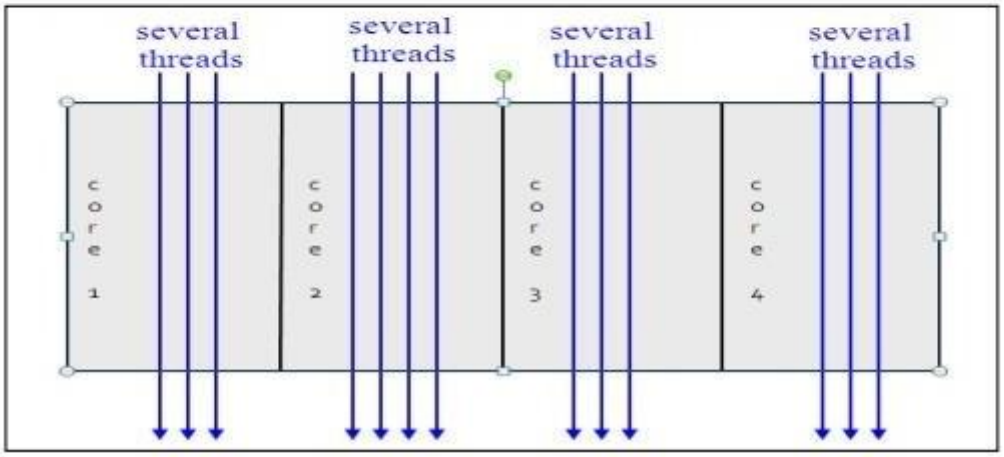
Motivation for Multi-core

- Exploits increased feature-size anddensity
- Increases functional units per chip (spatialefficiency)
- Limits energy consumption peroperation
- Constrains growth in processor complexity

➢ A multi-core processor is a processing system composed of two or more independent cores (orCPUs).Thecoresaretypicallyintegratedontoasingleintegratedcircuitdie(knownas a chip multiprocessor or CMP), or they may be integrated onto multiple dies in a single chippackage.

- **Amany-coreprocessor**isoneinwhichthenumberofcoresislargeenoughthattraditional multi-processor techniques are no longer efficient - this threshold is somewhere in the range of several tens of cores - and likely requires a network onchip.
- A multi-core processor implements multiprocessing in a single physical package. Cores in a multi-core device may be coupled together tightly or loosely. For example, cores mayor may not share caches, and they may implement message passing or shared memory inter-core communication methods. Common network topologies to interconnect coresinclude: bus, ring, 2-dimentional mesh, andcrossbar.
- All cores are identical in symmetric multi-core systems and they are not identical in asymmetric multi-core systems. Just as with single-processor systems, cores in multi-core systems may implement architectures such as superscalar, vector processing, or multithreading.
- Multi-coreprocessorsarewidelyusedacrossmanyapplicationdomainsincluding:general-purpose, embedded, network, digital signal processing, andgraphics.
- The amount of performance gained by the use of a multi-core processor is strongly dependent on the software algorithms andimplementation.
- Multi-coreprocessingisagrowingindustrytrendassinglecoreprocessorsrapidlyreachthe physical limits of possible complexity andspeed.
- Companiesthathaveproducedorareworkingonmulti-coreproductsincludeAMD,ARM, Broadcom, Intel, andVIA.
- withasharedon-chipcachememory,communicationeventscanbereducedtojustahandful of processorcycles.
- therefore with low latencies, communication delays have a much smaller impact on overall performance.
- threads can also be much smaller and still beeffective
- automatic parallelization more feasible.





Multiple cores run in parallel

Properties of Multi-core systems

- Cores will be shared with a wide range of other applicationsdynamically.
- Load can no longer be considered symmetric across thecores.
- Coreswilllikelynotbeasymmetricasacceleratorsbecomecommonforscientifichardware.
- Sourcecodewilloftenbeunavailable,preventingcompilationagainstthespecifichardware configuration.

Applications that benefit from multi-core

- Databaseservers
- Webservers
- Telecommunicationmarkets
- Multimediaapplications
- Scientificapplications

In general, applications with Thread-level parallelism (as opposed to instruction-level parallelism

UNIT V MEMORY AND I/O SYSTEMS

MEMORY HIERARCHY

The entire computer memory can be viewed as the hierarchy depicted in Figure 4.13. The fastest access is to data held in processor registers. Therefore, if we consider the registers to be part of the memory hierarchy, then the processor registers are at the top in terms of the speed of access. The registers provide only a minuscule portion of the required memory.

At the next level of the hierarchy is a relatively small amount of memory that can be implemented directly on the processor chip. This memory, called a processor cache, holds copies ofinstructionsanddatastoredinamuchlargermemorythatisprovidedexternally.Thereareoften two levels ofcaches.

A primary cache is always located on the processor chip. This cache is small because it competes for space on the processor chip, which must implement many other functions. The primary cache is referred to as level (L1) cache. A larger, secondary cache is placed between the primary cache and the rest of the memory. It is referred to as level 2 (L2) cache. It is usually implemented using SRAM chips. It is possible to have both Ll and L2 caches on the processor chip.

The next level in the hierarchy is called the main memory. This rather large memory is implemented using dynamic memory components, typically in the form of SIMMs, DIMMs, or RIMMs. The main memory is much larger but significantly slower than the cache memory. In a typical computer, the access time for the main memory is about ten times longer than the access time for the L 1 cache.

Diskdevicesprovideahugeamountofinexpensivestorage.Theyareveryslowcompared to the semiconductor devices used to implement the main memory. A hard disk drive (HDD; also hard drive, hard disk, magnetic disk or disk drive) is a device for storing and retrieving digital information, primarily computer data. It consists of one or more rigid (hence "hard") rapidly rotatingdiscs(oftenreferredtoasplatters),coatedwithmagneticmaterialandwithmagneticheads arranged to write data to the surfaces and read it fromthem.

During program execution, the speed of memory access is of utmost importance. The key to managing the operation of the hierarchical memory system in is to bring the instructions and

data that will be used in the near future as close to the processor as possible. This can be done by using the hardwaremechanisms.

MEMORYTECHNOLOGIES

Memory latency is traditionally quoted using two measures—access time and cycle time. Accesstimeisthetimebetweenwhenareadisrequestedandwhenthedesiredwordarrives,cycle timeistheminimumtimebetweenrequeststomemory.Onereasonthatcycletimeisgreaterthan access time is that the memory needs the address lines to be stable betweenaccesses.

DRAM technology

The main memory of virtually every desktop or server computer sold since 1975 is composed of semiconductor DRAMs,. As early DRAMs grew in capacity, the cost of a package with all the necessary address lines was an issue. The solution was to multiplex the

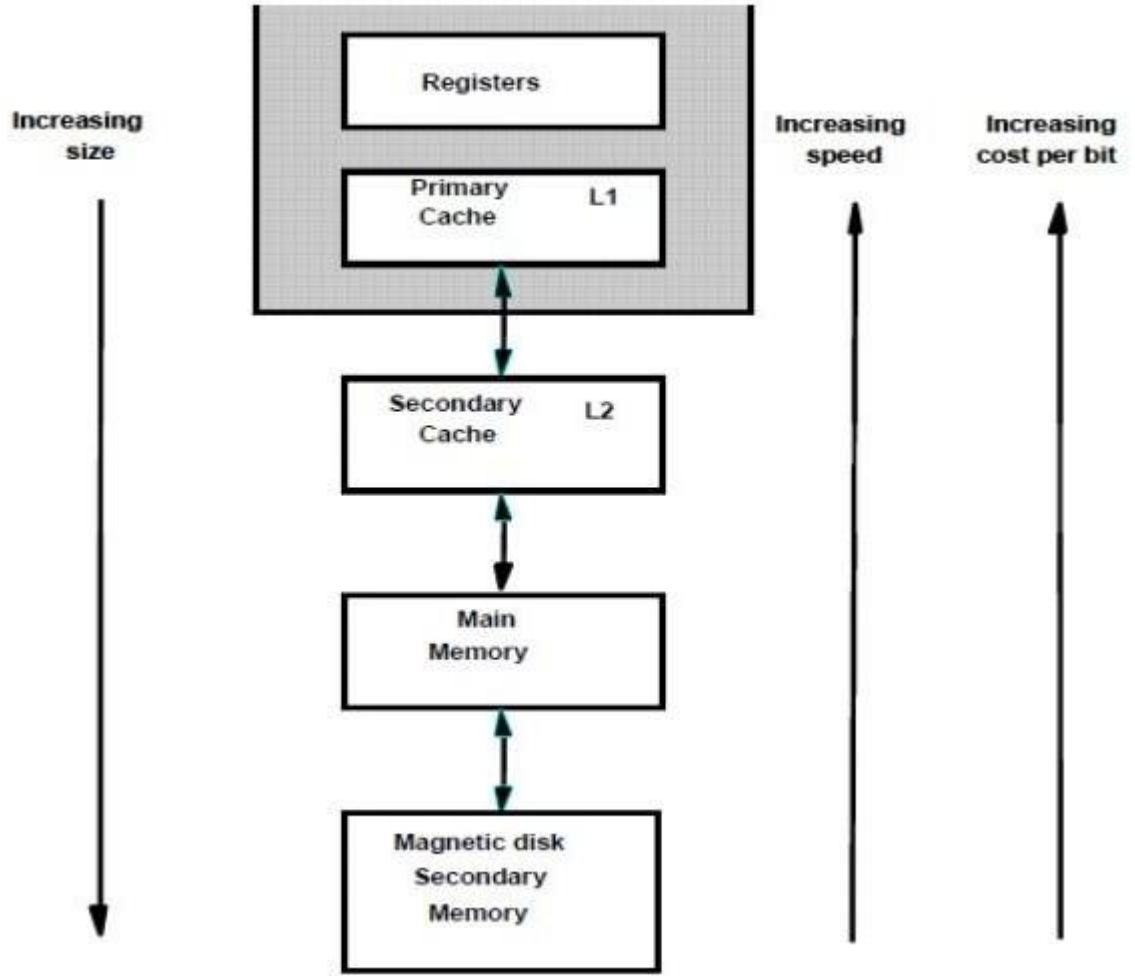


Fig5.1 DRAM technology

address lines, thereby cutting the number of address pins in half. One half of the address is sent first, called the row access strobe or(RAS). It is followed by the other half of the address, sent during the column access strobe(CAS). These names come from the internal chip organization, since the memory is organized as a rectangular matrix addressed by rows and columns.

DRAMs are commonly sold on small boards called DIMMs for Dual Inline Memory Modules.DIMMstypicallycontain4to16DRAMs.Theyarenormallyorganizedtobeeightbytes wide for desktopsystems.

SRAM Technology

IncontrasttoDRAMsareSRAMs—thefirstletterstandingforstatic.Thedynamicnature of the circuits in DRAM require data to be written back after being read, hence the difference betweentheaccesstimeandthecycletimeaswellastheneedtorefresh.SRAMstypicallyusesix transistors per bit to prevent the information from being disturbed whenread.

In DRAM designs the emphasis is on cost per bit and capacity, while SRAM designs are concerned with speed and capacity. (Because of this concern, SRAM address lines are not multiplexed.). Thus, unlike DRAMs, there is no difference between access time and cycle time. For memories designed in comparable technologies, the capacity of DRAMs is roughly 4 to 8 timesthatofSRAMs.Thecycletimeof SRAMsis8to16timesfasterthan DRAMs,buttheyare also 8 to 16 times asexpensive.

Embedded Processor Memory Technology: ROM and Flash

Embedded computers usually have small memories, and most do not have a disk to act as non-volatile storage. Two memory technologies are found in embedded computers to addressthis problem.

The first is Read-Only Memory (ROM). ROM is programmed at time of manufacture, needingonlyasingletransistorperbittorepresent1or0.ROMisusedfortheembeddedprogram andforconstants,oftenincludedaspartofalargerchip.Inadditiontobeingnon-volatile,ROMis alsonon-destructible;nothingthecomputercandocanmodifythecontentsofthismemory.Hence, ROM also provides a level of protection to the code of embedded computers. Since addressbased protection is often not enabled in embedded processors, ROM can fulfill an importantrole.

The second memory technology offers non-volatility but allows the memory to be modified.Flashmemoryallowstheembeddeddevicetoalternonvolatilememoryafterthesystem is manufactured, which can shorten productdevelopment.

Improving Memory Performance in a standard DRAM Chip

To improve bandwidth, there have been a variety of evolutionary innovations over time.

1. Thefirstwastimingsignalsthatallowrepeatedaccessestotherowbufferwithoutanother row access time, typically called fast pagemode..

2. The second major change is that conventional DRAMs have an asynchronousinterface to the memory controller, and hence every transfer involves overhead to synchronize with the controller.This optimization is called Synchronous DRAM(SDRAM).

3. ThethirdmajorDRAMinnovationtoincreasebandwidthistotransferdataonboththe rising edge and falling edge of the DRAM clock signal, thereby doubling the peak data rate. This optimization is calledDoubleDataRate(DDR).

CACHEBASICS

Basic Ideas

The cache is a small mirror-image of a portion (several "lines") of main memory. cache is faster than main memory ==> so maximize its utilization

- cache is more expensive than main memory ==> so it is muchsmaller

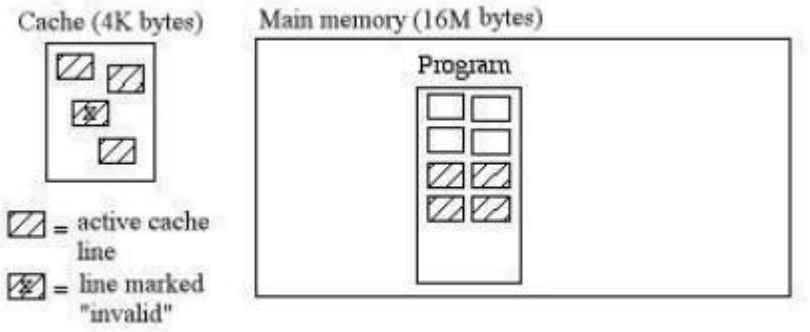
Locality of reference

Theprinciplethattheinstructioncurrentlybeingfetched/executedisverycloseinmemory to the instruction to be fetched/executed next. The same idea applies to the data value currently being accessed (read/written) inmemory.

If we keep the most active segments of program and data in the cache, overall execution speed for the program will be optimized. Our strategy for cache utilization should maximize the numberofcacheread/writeoperations,incomparisonwiththenumberofmainmemoryread/write operations.

Example

A line is an adjacent series of bytes in main memory (that is, their addresses are contiguous). Suppose a line is 16 bytes in size. For example, suppose we have a 2^{12}= 4K-byte cache with 2^8 = 256 16-byte lines; a 2^{24} = 16M-byte main memory, which is 2^{12} = 4K times the size of the cache; and a 400-line program, which will not all fit into the cache at once.



Each active cache line is established as a copy of a corresponding memory line during execution. Whenever a memory write takes place in the cache, the "Valid" bit is reset (marking that line "Invalid"), which means that it is no longer an exact image of its corresponding line in memory.

Cache Dynamics

When a memory read (or fetch) is issued by the CPU:

1. If the line with that memory address is in the cache (this is called a cache hit), the data is read from the cache to theMDR.

2. If the line with that memory address is not in the cache (this is called a miss), the cache is updated by replacing one of its active lines by the line with that memory address, and then the data is read from the cache to theMDR.

When a memory write is issued by the CPU:

1. If the line with that memory address is in the cache, the data is written from the MDR to the cache, and the line is marked "invalid" (since it no longer is an image of the corresponding memoryline
2. Ifthelinewiththatmemoryaddressisnotinthecache,thecacheisupdatedbyreplacing one of its active lines by the line with that memory address. The data is then written from the MDR to the cache and the line is marked"invalid."

Cache updating is done in the following way.

1. A candidate line is chosen for replacement using an algorithm that tries to minimize the numberofcacheupdatesthroughoutthelifeoftheprogramrun.Twoalgorithmshavebeen popular in recentarchitectures:

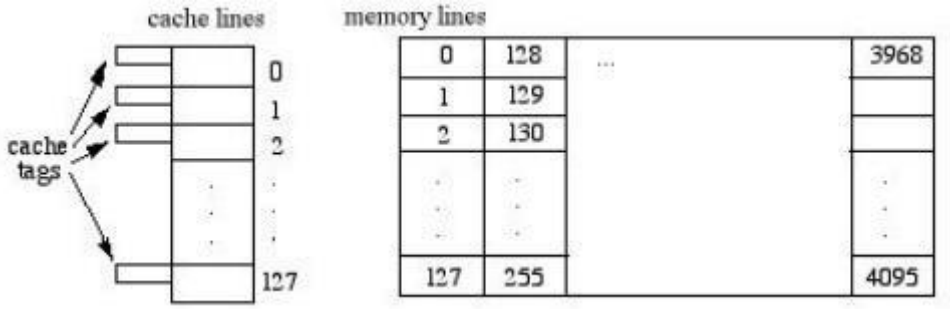
   - Choose the line that has been least recently used - "LRU" for short (e.g., thePowerPC)

- Choose the line randomly (e.g., the68040)

2. If the candidate line is "invalid," write out a copy of that line to main memory (thus bringing the memory up to date with all recent writes to that line in thecache).
3. Replace the candidate line by the new line in thecache.

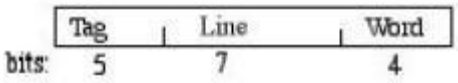
MEASURING AND IMPROVING CACHE PERFORMANCE

As a working example, suppose the cache has 2^7 = 128 lines, each with 2^4 = 16 words. Suppose the memory has a 16-bit address, so that 2^{16} = 64K words are in the memory's address space.



Direct Mapping

Under this mapping scheme, each memory line j maps to cache line j mod 128 so the memory address looks like this:



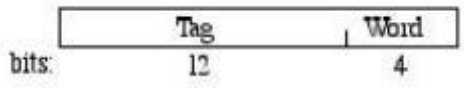
Here, the "Word" field selects one from among the 16 addressable words in a line. The "Line" field defines the cache line where this memory line should reside. The "Tag" field of the address is is then compared with that cache line's 5-bit tag to determine whether there is a hit or a miss.Ifthere'samiss,weneedtoswapoutthememorylinethatoccupiesthatpositioninthecache and replace it with the desired memoryline.

E.g., Suppose we want to read or write a word at the address 357A, whose 16 bits are 0011010101111010. This translates to Tag = 6, line = 87, and Word = 10 (all in decimal). If line 87inthecachehasthesametag(6),thenmemoryaddress357Aisinthecache.Otherwise,amiss hasoccurredandthecontentsofcacheline87mustbereplacedbythememoryline001101010111 = 855 before the read or write is executed.

Directmappingisthemostefficientcachemappingscheme,butitisalsotheleasteffective in its utilization of the cache - that is, it may leave some cache linesunused.

Associative Mapping

This mapping scheme attempts to improve cache utilization, but at the expense of speed. Here,thecachelinetagsare12bits,ratherthan5,andanymemorylinecanbestoredinanycache line. The memory address looks likethis:



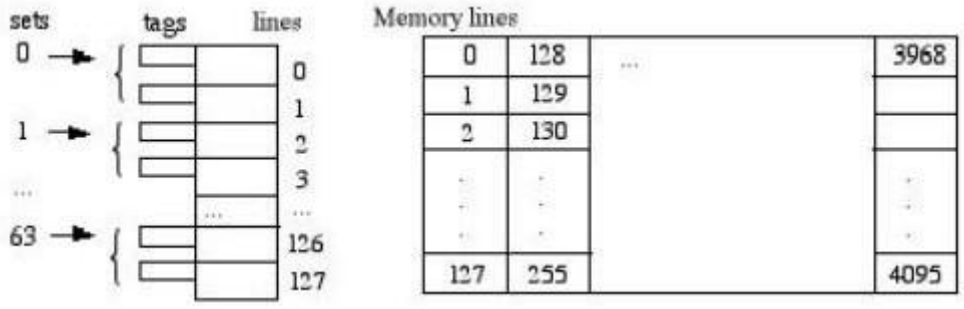
Here, the "Tag" field identifies one of the $2^{12} = 4096$ memory lines; all the cache tags are searched to find out whether or not the Tag field matches one of the cache tags. If so, we have a hit,andifnotthere'samissandweneedtoreplaceoneofthecachelinesbythislinebeforereading or writing into the cache. (The "Word" field again selects one from among 16 addressable words (bytes) within theline.)

For example, suppose again that we want to read or write a word at the address 357A, whose 16 bits are 0011010101111010. Under associative mapping, this translates to Tag = 855 and Word = 10 (in decimal). So we search all of the 128 cache tags to see if any one of them will match with 855. If not, there's a miss and we need to replace one of the cache lines with line 855 from memory before completing the read or write.

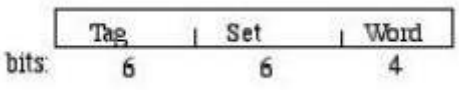
The search of all 128 tags in the cache is time-consuming. However, the cache is fully utilizedsincenoneofitslineswillbeunusedpriortoamiss(recallthatdirectmappingmaydetect a miss even though the cache is not completely full of activelines).

Set-associative Mapping

Thisschemeisacompromisebetweenthedirectandassociativeschemesdescribedabove. Here,thecacheisdividedintosetsoftags,andthesetnumberisdirectlymappedfromthememory address(e.g.,memorylinejismappedtocachesetjmod64),assuggestedbythediagrambelow:



The memory address is now partitioned to like this:



Here, the "Tag" field identifies one of the $2^6 = 64$ different memory lines in each of the 2^6=64different"Set"values.Sinceeachcachesethasroomforonlytwolinesatatime,thesearch for a match is limited to those two lines (rather than the entire cache). If there's a match, we have a hit and the read or write can proceedimmediately.

Otherwise, there's a miss and we need to replace one of the two cache lines by this line before reading or writing into the cache. (The "Word" field again select one from among 16 addressablewordsinsidetheline.) In set-associativemapping,whenthenumberoflinespersetis n, the mapping is called n-way associative. For instance, the above example is 2-wayassociative.

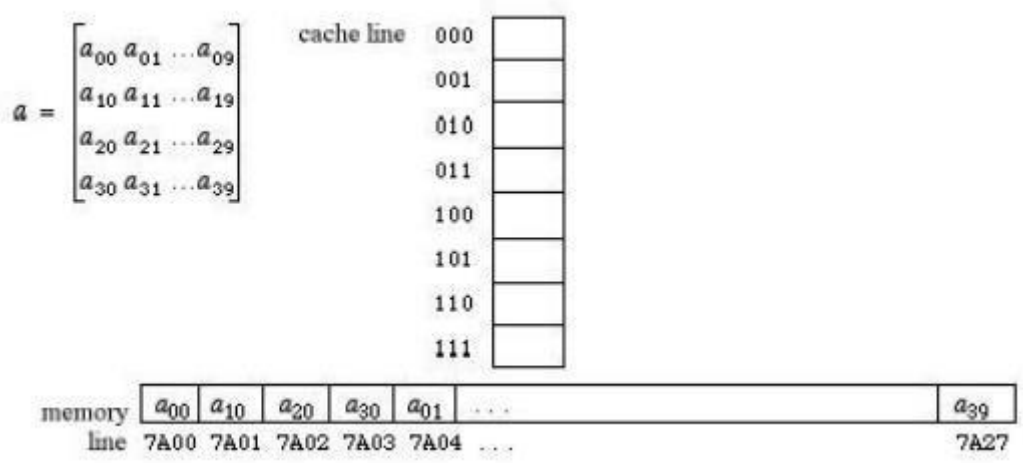
E.g., Again suppose we want to read or write a word at the memory address 357A, whose 16 bits are 0011010101111010. Under set-associative mapping, this translates to Tag = 13, Set = 23, and Word = 10 (all in decimal). So we search only the two tags in cache set 23 to see if either one matches tag 13. If so, we have a hit. Otherwise, one of these two must be replaced by the memory line being addressed (good old line 855) before the read or write can be executed.

A Detailed Example

Supposewehavean8-wordcacheanda16-bitmemoryaddressspace,whereeachmemory "line" is a single word (so the memory address need not have a "Word" field to distinguish individualwordswithinaline).Supposewealsohavea4x10arrayaofnumbers(onenumberper addressible memory word) allocated in memory column-by-column, beginning at address 7A00. That is, we have the following declaration and memory allocation picturefor

The array a:

float [a = new float [4][10];



Here is a simple equation that recalculates the elements of the first row ofa:

$$a_{0,i} = \frac{a_{0,i}}{\sum_{j=0}^{9} a_{0,j}/10} \qquad \text{for all } i = 0, 1, \ldots, 9$$

This calculation could have been implemented directly in C/C++/Java asfollows:

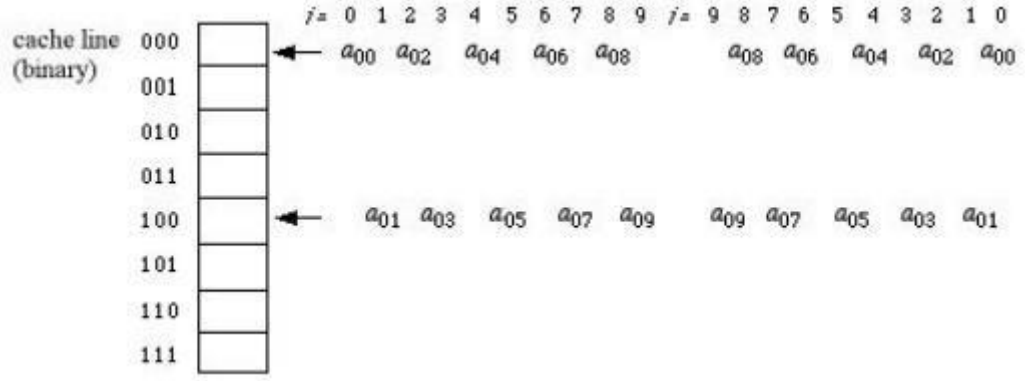
```
Sum = 0;
for (j=0; j<=9; j++)
   Sum = Sum + a[0][j];
Ave = Sum / 10;
for (i=9; i>=0; i--)
   a[0][i] = a[0][i] / Ave;
```

Theemphasishereisontheunderlinedpartsofthisprogramwhichrepresentmemoryread andwriteoperationsinthearraya.Notethatthe3rdand6thlinesinvolveamemoryreadofa[0][j] anda[0][i],andthe6thlineinvolvesamemorywriteofa[0][i].Soaltogether,thereare20memory reads and 10 memory writes during the execution of this program. The following discussion focusses on those particular parts of this program and their impact on thecache.

Direct Mapping

Direct mapping of the cache for this model can be accomplished by using the rightmost 3 bitsofthememoryaddress.Forinstance,thememoryaddress7A00=0111101000000000,which maps to cache address 000. Thus, the cache address of any value in the array a is just its memory address modulo8.

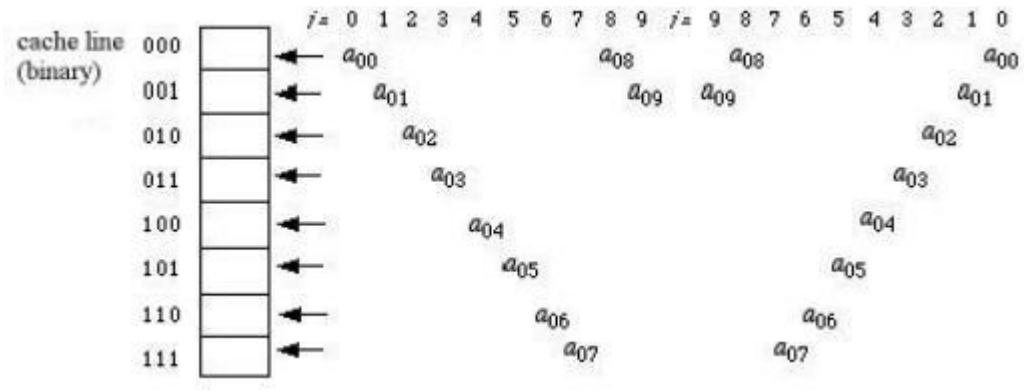
Using this scheme, we see that the above calculation uses only cache words 000 and 100, since each entry in the first row of a has a memory address with either 000 or 100 as its rightmost 3bits.Thehitrateofaprogramisthenumberofcachehitsamongitsreads andwritesdividedby the total number of memory reads and writes. There are 30 memory reads and writes for this program,andthefollowingdiagramillustratescacheutilizationfordirectmappingthroughoutthe life of these twoloops:



Reading the sequence of events from left to right over the ranges of the indexes i and j, it iseasytopickoutthehitsandmisses.Infact,thefirstloophasaseriesof10misses(nohits).The secondloopcontainsareadandawriteofthesamememorylocationoneachrepetition(i.e.,a[0][i] = a[0][i]/Ave;), so that the 10 writes are guaranteed to be hits. Moreover, the first two repetitions of the second loop have hits in their read operations, since a_{09} and a_{08} are still in the cache at the end of the first loop. Thus, the hit rate for direct mapping in this algorithm is 12/30 = 40%

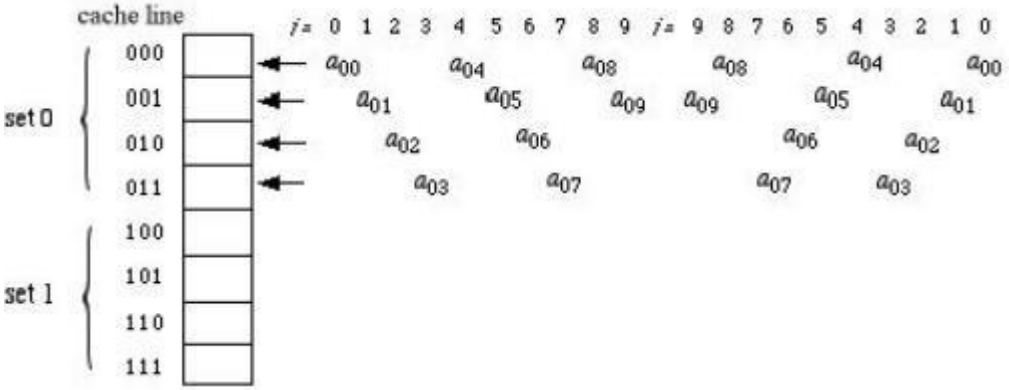
Associative Mapping

Associativemappingforthisproblemsimplyusestheentireaddressasthecachetag.Ifwe use the least recently used cache replacement strategy, the sequence of events in the cache after thefirstloopcompletesisshowninthe left-halfofthefollowingdiagram.Thesecondloophappilyfinds all of a $_{09}$ - a_{02} already in the cache, so it will experience a series of 16 hits (2 for each repetition)beforemissingona$_{01}$wheni=1.Thelasttwostepsofthesecondloopthereforehave2 hits and 2misses.



Set-Associative Mapping

Set associative mapping tries to compromise these two. Suppose we divide the cache into two sets, distinguished from each other by the rightmost bit of the memory address, and assume the least recently used strategy for cache line replacement. Cache utilization for our program can now be pictured as follows:



Hereallthe entries in a that are referenced in this algorithm have even-numbered addresses (their rightmost bit=0), so only the top half of the cache is utilized. The hit rate is therefore slightly worse than associative mapping and slightly better than direct. That is, set-associative cache mapping for this program yields 14 hits out of 30 read/writes for a hit rate of 46%.

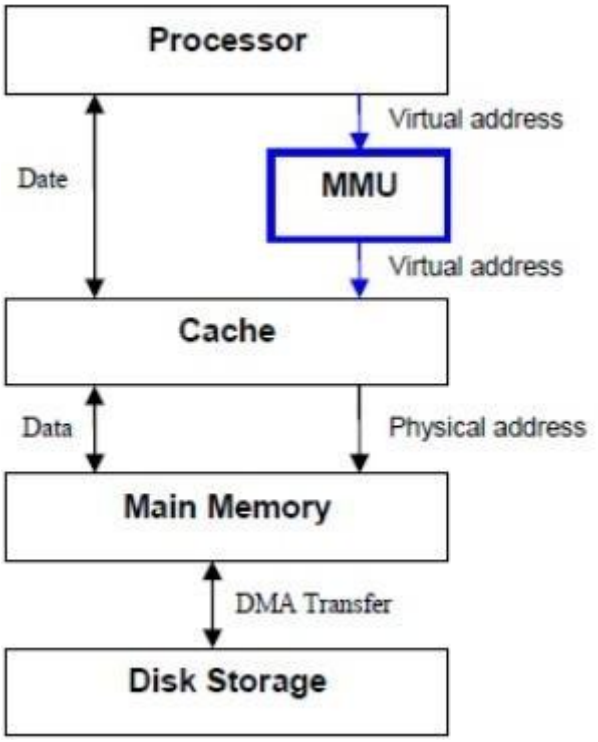
VIRTUAL MEMORY

The physical main memory is not as large as the address space spanned by an address issued by the processor. When a program does not completely fit into the main memory, the parts of it not currently being executed are stored on secondary storage devices, such as magnetic disks. Of course, all parts of a program that are eventually executed are first brought into the main memory.

When a new segment of a program is to be moved into a full memory, it must replace another segment already in the memory. The operating system moves programs and data automatically between the main memory and secondary storage. This process is known as swapping. Thus, the application programmer does not need to be aware of limitations imposedby the available mainmemory.

Techniques that automatically move program and data blocks into the physical main memory when they are required for execution are called virtual-memory techniques. Programs, andhencetheprocessor,referenceaninstructionanddataspacethatisindependentoftheavailable physicalmainmemoryspace.Thebinaryaddressesthattheprocessorissuesforeitherinstructions or data are called virtual or logical addresses. These addresses are translated into physical addresses by a combination of hardware and software components. If a virtual address refers to a partoftheprogramordataspacethatiscurrentlyinthephysicalmemory,thenthecontentsofthe appropriate location in the main memory are accessed immediately. On the other hand, if the referencedaddressisnotinthemainmemory,itscontentsmustbebroughtintoasuitablelocation in the memory before they can beused.

Figure shows a typical organization that implements virtual memory. A special hardware unit, called the Memory Management Unit (MMU), translates virtual addresses into physical addresses. When the desired data (or instructions) are in the main memory, these data are fetched as described in our presentation of the ache mechanism. If the data are not in the main memory, theMMUcausestheoperatingsystemtobringthedataintothememoryfromthedisk.TheDMA scheme is used to perform the data Transfer between the disk and the mainmemory.



ADDRESS TRANSLATION

The process of translating a virtual address into physical address is known as address translation.ItcanbedonewiththehelpofMMU.Asimplemethodfortranslatingvirtualaddresses intophysicaladdressesistoassumethatallprogramsanddataarecomposedoffixed-lengthunits called *pages*, each of which consists of a block of words that occupy contiguous locations in the main memory. Pages commonly range from 2K to 16K bytes in length. They constitute the basic unitofinformationthatismovedbetweenthemainmemoryandthediskwheneverthetranslation mechanism determines that a move isrequired.

Pages should not be too small, because the access time of a magnetic disk is much longer (severalmilliseconds)thantheaccesstimeofthemainmemory.Thereasonforthisisthatittakes a considerable amount of time to locate the data on the disk, but once located, the data can be transferredatarateofseveralmegabytespersecond.Ontheotherhand,ifpagesaretoolargeitis possiblethatasubstantialportionofapagemaynotbeused,yetthisunnecessarydatawilloccupy valuable space in the mainmemory.

The cache bridges the speed gap between the processor and the main memory and is implemented in hardware. The virtual-memory mechanism bridges the size and speed gaps between the main memory and secondary storage and is usually implemented in part by software techniques.Conceptually,cachetechniquesandvirtual-memorytechniquesareverysimilar.They differ mainly in the details of theirimplementation.

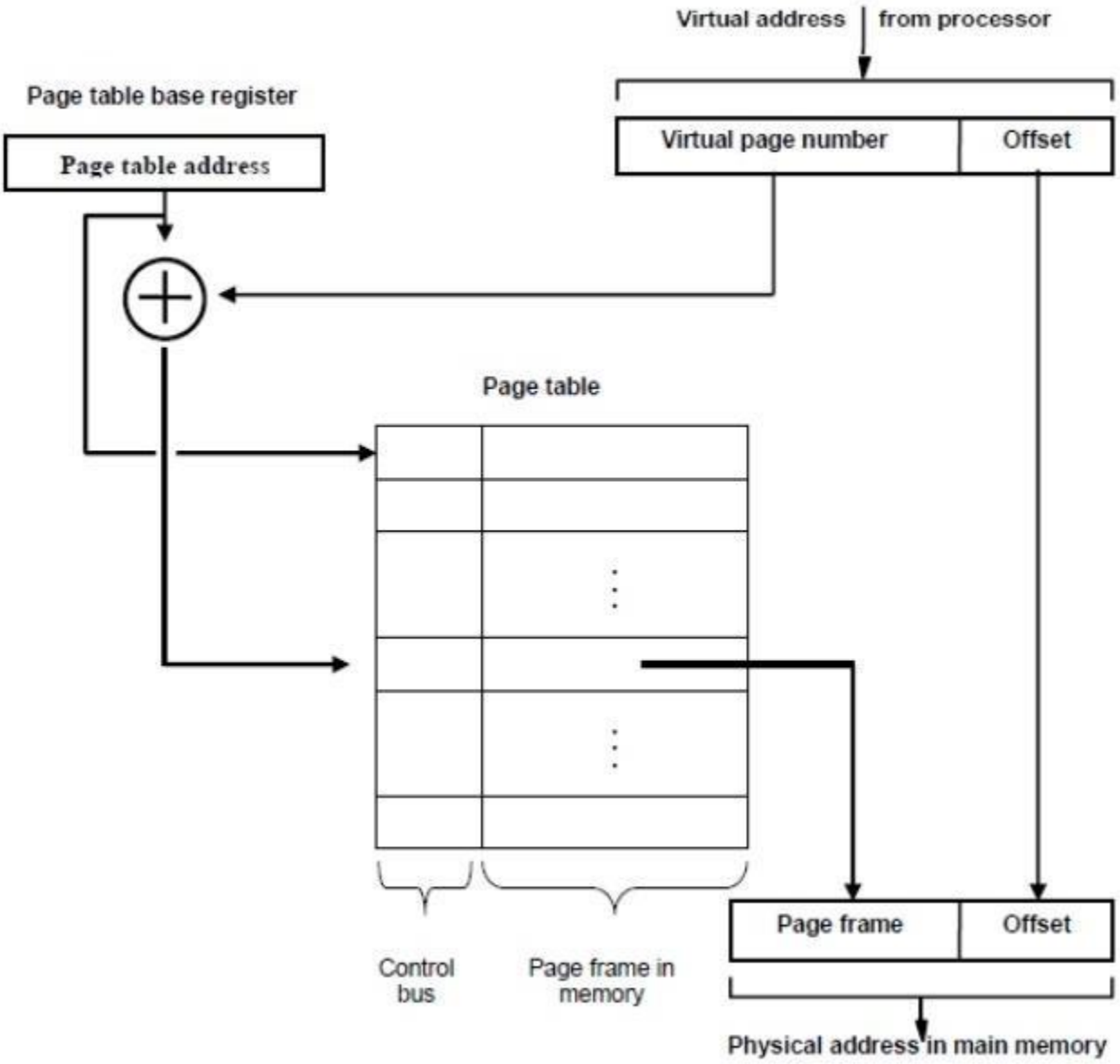
A virtual-memory address translation method based on the concept of fixed-length pages. Eachvirtualaddressgeneratedbytheprocessor,whetheritisforaninstructionfetchoranoperand fetch/store operation, is interpreted as a *virtual page number* (high-order bits) followed by an *offset* (low-order bits) that specifies the location of a particular byte (or word) within a page. Informationaboutthemainmemorylocationofeachpageiskeptinapagetable.Thisinformation includes the main memory address where the page is stored and the current status of thepage.

An area in the main memory that can hold one page is called a *page frame*. The starting address of the page table is kept in a *page table base register*. By adding the virtual page number tothecontentsofthisregister,theaddressofthecorrespondingentryinthepagetableisobtained. The contents of this location give the starting address of the page if that page currently resides in the main memory. Each entry in the page table also includes some control bits that describe the status of the page while it is in the main memory. One bit indicates the validity of the page, that is, whether the page is actually loaded in the mainmemory.

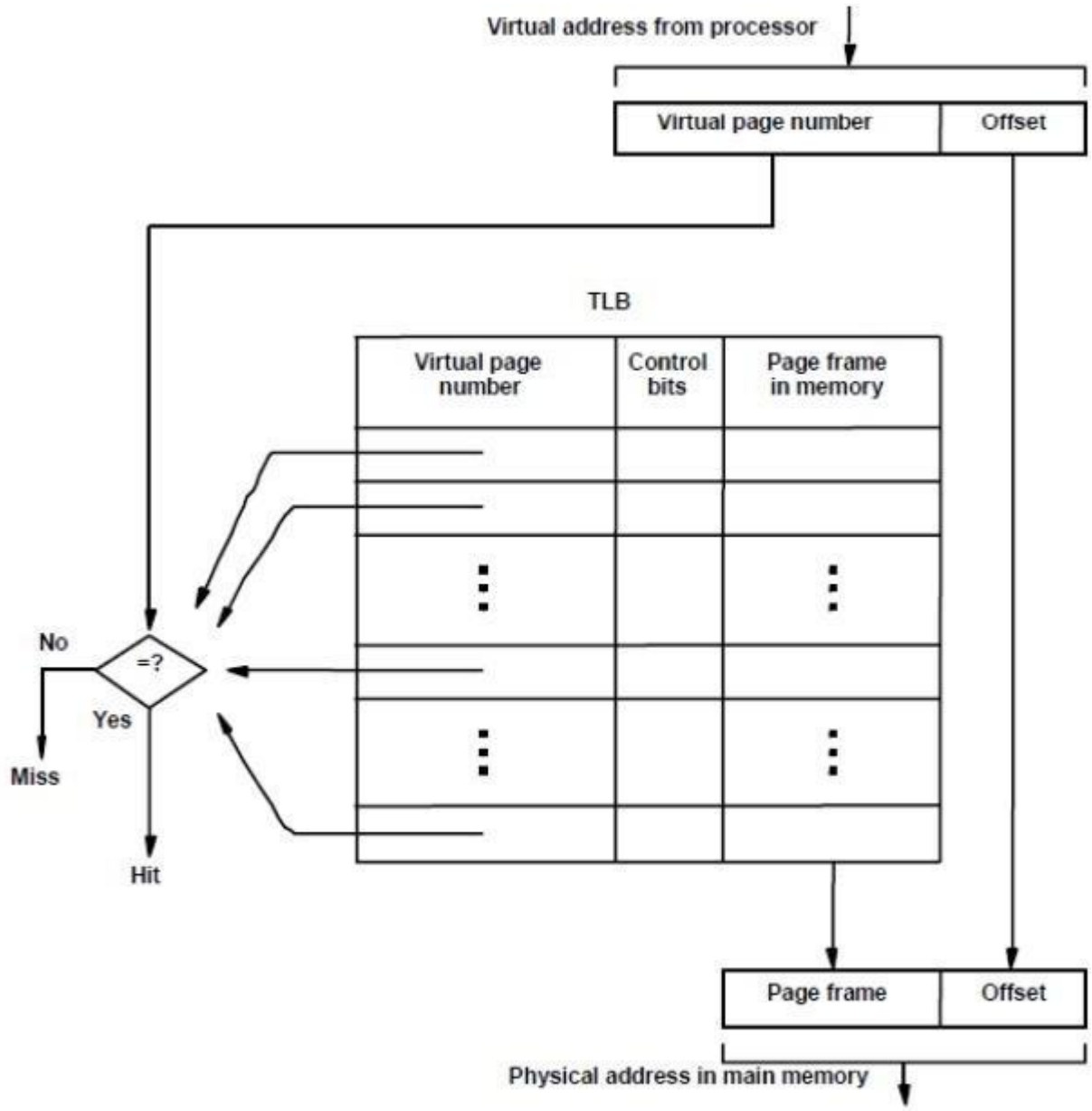
This bit allows the operating system to invalidate the page without actually removing it. Another bit indicates whether the page has been modified during its residency in the memory. As in cache memories, this information is needed to determine whether the page should be written backtothediskbeforeitisremovedfromthemainmemorytomakeroomforanotherpage.Othercontrolbit sindicatevariousrestrictionsthatmaybeimposedonaccessingthepage.Forexample, a program may be given full read and write permission, or it may be restricted to read accesses only.

TLBS- INPUT/OUTPUT SYSTEM

TheMMUmustusethepagetableinformationforeveryreadandwriteaccess;soideally, the page table should be situated within the MMU. Unfortunately, the page table may be rather large, and since the MMU is normally implemented as part of the processor chip (along with the primary cache), it is impossible to include a complete page table on this chip. Therefore, the page table is kept in the main memory. However, a copy of a small portion of the page table can be accommodated within the MMU.



Thisportionconsistsofthepagetableentriesthatcorrespondtothemostrecentlyaccessedpages. A small cache, usually called the *Translation Lookaside Buffer*(TLB) is incorporated into the MMUforthispurpose.TheoperationoftheTLBwithrespecttothepagetableinthemainmemory is essentially the same as the operation of cache memory; the TLB must also include the virtual addressoftheentry.FigureshowsapossibleorganizationofaTLBwheretheassociative-mapping technique is used. Setassociative mapped TLBs are also found in commercialproducts.



An essential requirement is that the contents of the TLB be coherent with the contents of pagetablesinthememory.Whentheoperatingsystemchangesthecontentsofpagetables,itmust simultaneouslyinvalidatethecorrespondingentriesintheTLB.OneofthecontrolbitsintheTLB is provided for this purpose. When an entry is invalidated, the TLB will acquire the new information as part ofthe MMU's normal response to access misses. Address translation proceeds as follows.

Given a virtual address, the MMU looks in the TLB for the referenced page. IT the page tableentryforthispageisfoundintheTLB,thephysicaladdressisobtainedimmediately.Ifthere is a miss in the TLB, then the required entryis obtained from the page table in the main memory

and the TLB is updated. When a program generates an access request to a page that is not in the main memory, a *page fault* is said to have occurred. The whole page must be brought from the disk into the memory before access can proceed. When it detects a page fault, the MMU asks the operating system to intervene by raising an exception (interrupt). Processing of the active task is interrupted, and control is transferred to the operating system.

The operating system then copies the requested page from the disk into the main memory and returns control to the interrupted task. Because a long delay occurs while the page transfer takes place, the operating system may suspend execution of the task that caused the page fault and begin execution of another task whose pages are in the main memory.

It is essential to ensure that the interrupted task can continue correctly when it resumes execution. A page fault occurs when some instruction accesses a memory operand that is not in the main memory, resulting in an interruption before the execution of this instruction is completed. Hence, when the task resumes, either the execution of the interrupted instruction must continue from the point of interruption, or the instruction must be restarted. The design of a particular processor dictates which of these options should be used.

If a new page is brought from the disk when the main memory is full, it must replace one of the resident pages. The problem of choosing which page to remove is just as critical here as it is in a cache, and the idea that programs spend most of their time in a few localized areas also applies. Because main memories are considerably larger than cache memories, it should be possible to keep relatively larger portions of a program in the main memory. This will reduce the frequency of transfers to and from the disk.

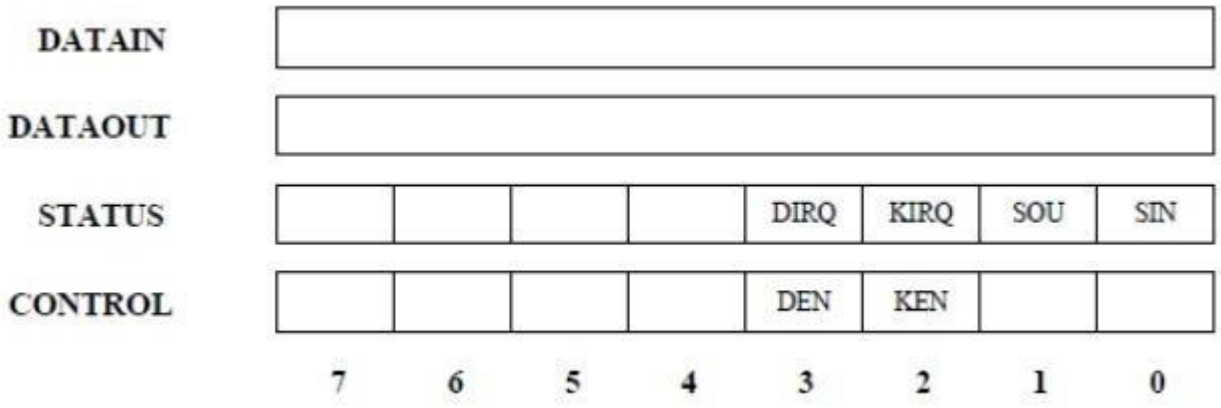
Concepts similar to the FIFO, Optimal and LRU replacement algorithms can be applied to page replacement and the control bits in the page table entries can indicate usage. One simple scheme is based on a control bit that is set to 1 whenever the corresponding page is referenced (accessed). The operating system occasionally clears this bit in all page table entries, thus providing a simple way of determining which pages have not been used recently.

A modified page has to be written back to the disk before it is removed from the main memory. It is important to note that the write-through protocol, which is useful in the framework of cache memories, is not suitable for virtual memory. The access time of the disk is so long that it does not make sense to access it frequently to write small amounts of data. The address translation process in the MMU requires some time to perform, mostly dependent on the time needed to look up entries in the TLB. Because of locality of reference, it is likely that many successive translations involve addresses on the same page. This is particularly evident in fetching instructions. Thus, we can reduce the average translation time by including one or more special registers that retain the virtual page number and the physical page frame of the most recently

performed translations. The information in these registers can be accessed more quickly than the TLB.

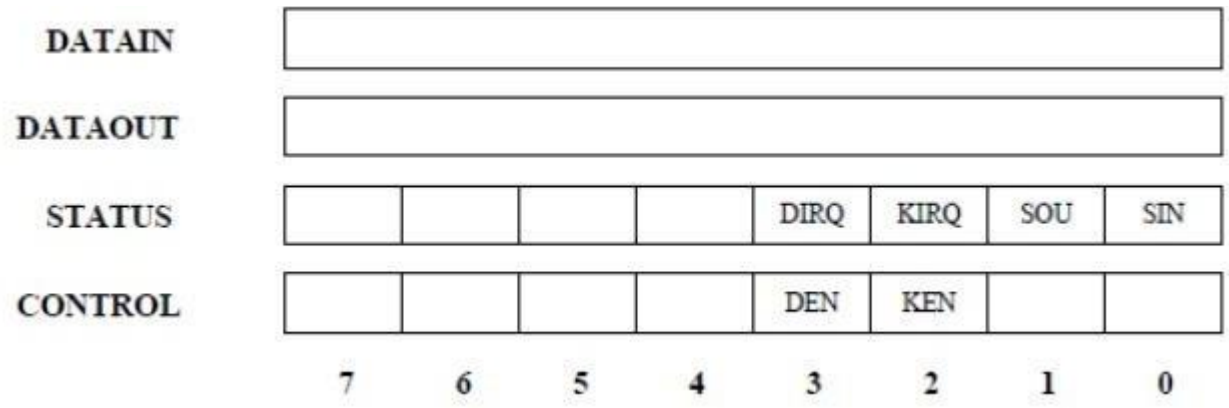
PROGRAMMED I/O

ConsiderasimpleexampleofI/Ooperationsinvolvingakeyboardandadisplaydevicein acomputersystem.Thefourregistersshowninfigure5.3areusedinthedatatransferoperations. Register STATUS contains two control flags, SIN and SOUT, which provide status information forthekeyboardandthedisplayunit,respectively.ThetwoflagsKIRQandDIRQinthisregister are used in conjunction with interrupts. They, and the KEN and DEN bitsin registerCONTROL,DatafromthekeyboardaremadeavailableintheDATAINregister,anddata sent to the display are stored in the DATAOUTregister.



Registers in keyboard and display interfaces

In program-controlled I/O the processor repeatedly checks a status flag to achieve the requiredsynchronizationbetweentheprocessorandaninputoroutputdevice.Theprocessorpolls the device. There are two other commonly used mechanisms for implementing I/O operations: interrupts and direct memory access. In the case of interrupts, synchronization is achieved by having the I/O device send a special signal over the bus whenever it is ready for a data transfer operation.Directmemoryaccessisatechniqueusedforhigh-speedI/Odevices.Itinvolveshaving thedeviceinterfacetransferdatadirectlytoorfromthememory,withoutcontinuousinvolvement by theprocessor.



DMA AND INTERRUPTS

A special control unit is provided to allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor. This approach is called direct memory access, or DMA.

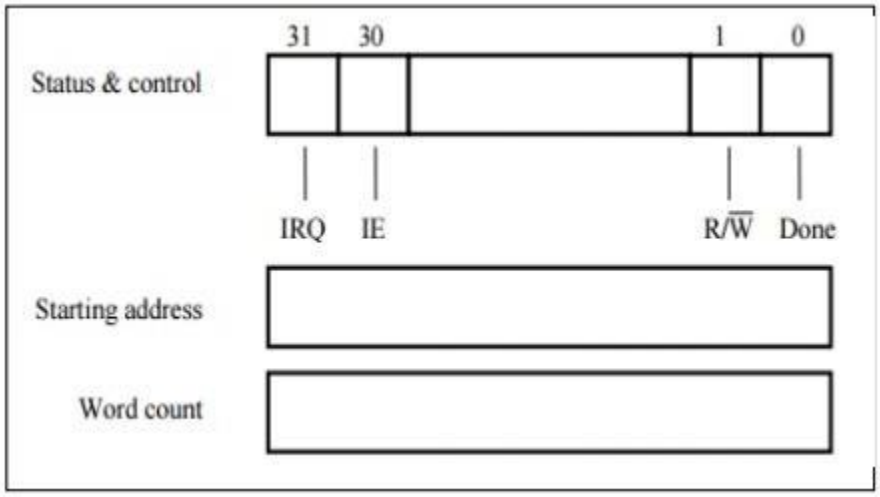
DMA transfers are performed by a control circuit that is part of the I/O device interface. We refer to this circuit as a DMA controller. The DMA controller performs the functions that would normally be carried out by the processor when accessing the main memory. For each word transferred, it provides the memory address and all the bus signals that control data transfer. Sinceithastotransferblocksofdata,theDMAcontrollermustincrementthememoryaddressfor successive words and keep track of the number oftransfers.

Although a DMA controller can transfer data without intervention by the processor, itsoperationmustbeunderthecontrolofaprogramexecutedbytheprocessor.Toinitiatethetransfer of a block of words, the processor sends the starting address, the number of words in the block, and the direction of the transfer. On receiving this information, the DMA controller proceeds to performtherequestedoperation.Whentheentireblockhasbeentransferred,thecontrollerinforms the processor by raising an interruptsignal.

While a DMA transfer is taking place, the program that requested the transfer cannot continue, and the processor can be used to execute another program. After the DMA transfer is completed, the processor can return to the program that requested the transfer. I/O operations are always performed by the operating system of the computer in response to a request from an application program.

The OS is also responsible for suspending the execution of one program and starting another. Thus, for an I/O operation involving DMA, the OS puts the program that requested the transfer in the Blocked state, initiates the DMA operation, and starts the execution of another

program. When the transfer is completed, the DMA controller informs the processor by sending an interrupt request. In response, the OS puts the suspended program in the Runnable state sothat it can be selected by the scheduler to continueexecution.



The above figure shows an example of the DMA controller registers that are accessed by the processor to initiate transfer operations. Two registers are used for storing the startingaddress and the word count. The third register contains status and control flags. The R/W bit determines the direction of the transfer. When this bit is set to 1 by a program instruction, the controller performs a read operation, that is, it transfers data from the memory to the I/O device. Otherwise, it performs a writeoperation.

When the controller has completed transferring a block of data and is ready to receive anothercommand,itsetstheDoneflagto1.Bit30istheInterrupt-enableflag,IE.Whenthisflag is set to 1, it causes the controller to raise an interrupt after it has completed transferring a block of data. Finally, the controller sets the IRQ bit to 1 when it has requested aninterrupt.

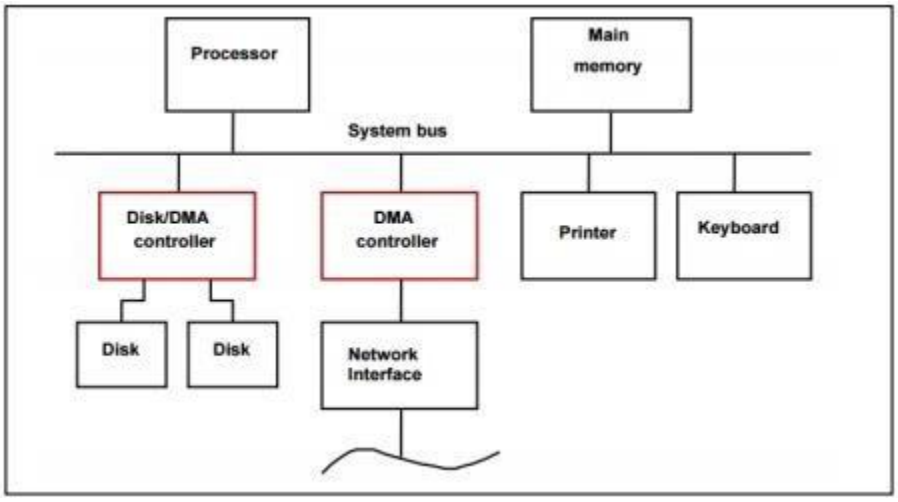
ADMAcontrollerconnectsahigh-speednetworktothecomputerbus.Thediskcontroller, which controls two disks, also has DMA capability and provides two DMA channels. It can perform two independent DMA operations, as if each disk had its own DMA controller. The registersneededtostorethememoryaddress,thewordcount,andsoonareduplicated,sothatone set can be used with each device.

To start a DMA transfer of a block of data from the main memory to one of the disks, a program writes the address and word count information into the registers of the corresponding channel of the disk controller. It also provides the disk controller with information to identify the data for future retrieval. The DMA controller proceeds independently to implement the specified operation.

WhentheDMAtransferiscompleted,thisfactisrecordedinthestatusandcontrolregister of the DMA channel by setting the Done bit. At the same time, if the IE bit is set, the controller sendsaninterruptrequesttotheprocessorandsetstheIRQbit.Thestatusregistercanalsobeused to record other information, such as whether the transfer took place correctly or errorsoccurred.

Memory accesses by the processor and the DMA controllers are interwoven. Requests by DMA devices for using the bus are always given higher priority than processor requests. Among differentDMAdevices,topprioritysgiventohigh-speedperipheralssuchasadisk,ahigh-speed networkinterface,oragraphicsdisplaydevice.Sincetheprocessororiginatesmostmemoryaccess cycles, the DMA controller can be said to "steal" memory cycles from the processor. Hence, this interweaving technique is usually called cyclestealing.

Alternatively, the DMA controller may be given exclusive access to the main memory to transfer a block of data without interruption. This is known as block or burst mode. Most DMA controllers incorporate a data storage buffer. In the case of the network interface for example, the DMA controller reads a block of data from the main memory and stores it into its input buffer. Thistransfertakesplaceusingburstmodeataspeedappropriatetothememoryandthecomputer bus. Then, the data in the buffer are transmitted over the network at the speed of thenetwork.



I/O PROCESSORS.

IO processor is

• A specializedprocessor

• Not only loads and stores into memory but also can execute instructions, which are among a set of I/O instructions

• The IOP interfaces to the system and devices

• ThesequenceofeventsinvolvedinI/Otransferstomoveoroperatetheresultsofan I/Ooperation into the main memory (using a program for IOP, which is also in mainmemory)
• Used to address the problem of direct transfer after executing the necessary format conversion or otherinstructions
• InanIOP-basedsystem,I/Odevicescandirectlyaccessthememorywithoutinterventionbythe processor

IOP instructions
• Instructions help in format conversions─ byte from memory as packed decimals to theoutput device forline-printer
• The I/O device data in different format can be transferred to main memory using anIOP

Sequence of events when using an I/O Processor
Sequence 1:

A DRQ (for IOP request) signal from an IOP device starts the IOP sequence, the IOP signals an interrupt on INTR line this requests attention from the processor

Sequence 2:
The processor responds by checking the device's status via the memory-mapped control registersandissuesacommandtellingtheIOPtoexecuteIOPinstructionsforthetransfertomove the formatted data into the memory.

Sequence 3:

During each successive formatted byte(s) transfer, the device DMAC (DMA controller) logic inside the IOP uses a processor bushold request line, HOLD, distinct from INTR device interrupt request line

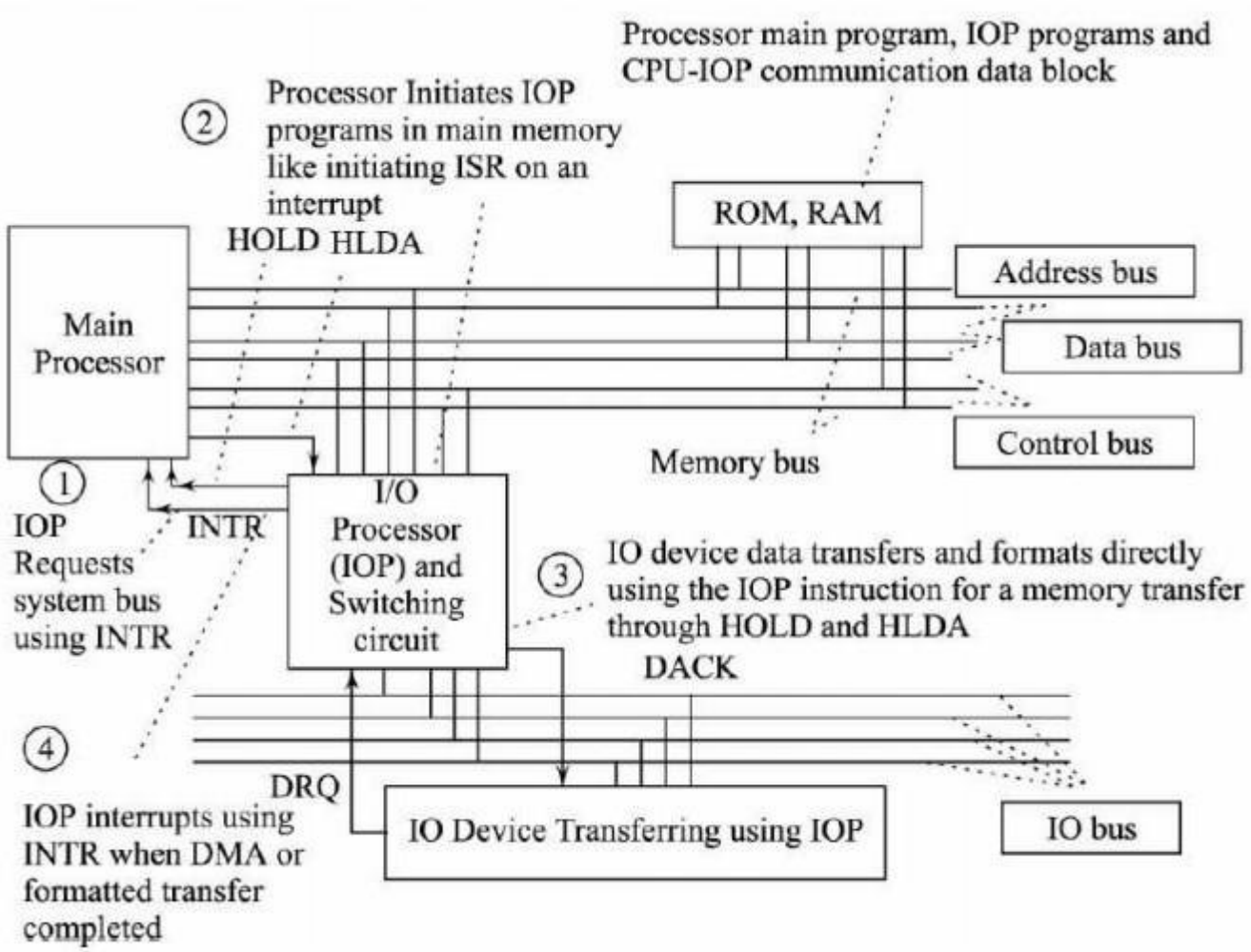
• The main processor sends to the device a signal from the processor called DACK (distinctfrom

INTA device-interrupt request-acknowledgeline)
• The I/O device bus has access to the address and data buses of the memory bus when DACKis activated

• It has no access when DACK is not activated when a HOLD request is not accepted by the processor when the processor is using the memorybus
• OncetheDMAlogicstartcommandhasbeenissuedtoIOP,themainprocessorbeginsworking on something else while the I/O device transfers the data into thememory

Sequence 4:

When the IOP's DMA transfer as per instructions is complete, the I/O device signals another interrupt (using DRQ) .Lets the main processor know that the DMA is done and it may access the data



Glossary
1BP: 1-bit branch predictor

4 C's - compulsory Misses: the first time a block is accessed by the cache **4 C's - capacity Misses:** blocks must be evicted due to the size of thecache.

4C's-coherenceMiss:processorsareaccessingthesameblock.ProcessorAwritestotheblock. EventhoughProcessorBhastheblockinitscache,itisamiss,becausetheblockisnolongerup- to-date.

4 C's - conflict Misses: associated with set associative and direct mapped caches - another data address needs the cache block and must replace the data currently in thecache.

ALAT: advance load table - stores advance information about load operations
aliasing:intheBTB,whentwoaddressesoverlapwiththesameBTBentry,thisiscalledaliasing. Aliasing should be kept to<1%.

ALU: arithmetic logic unit
AMAT: average memory access time
AMAT: Average Memory Access Time = hit time + miss rate * miss penalty
Amdahl'sLaw:anequationtodeterminetheimprovementofasystemwhenonlyaportionofthe system isimproved.

architectural registers: registers (Floating point and General Purpose) that are visible to theprogrammer.

ARF: architectural register file or retirement register file
Asynchronous Message Passing: a processor requests data, then continues processing instructions while message isretrieved.

BHT: branch history table - records if branch was taken or not taken.
blocking cache: the cache services only one block at a time, blocking all other requests **BTB:** branch target buffer - keeps track of what address was taken last timethe processor encountered thisinstruction.

cache coherence definition #1: Definition #1 - A read R from address X on processor P1 returns the value written by the most recent write W to X on P1 if no other processor has written to X between W and R.

cachecoherencedefinition#2: Definition#2- IfP1writestoXandP2readsXafterasufficient time, and there are no other writes to X in between, P2's read returns the value written by P1's write.

cache coherence definition #3: Definition #3 - Writes to the same location are serialized:two writes to location X are seen in the same order by allprocessors.

cachehit:desireddataisinthecacheandisup-to-date**cachemiss:**desireddataisnotinthecache or is dirty
cache thrashing: when two or more addresses are competing for the same cache block. The processor is requesting both addresses, which results in each access evicting the previous access. **CDB:** common databus

check pointing: store the state of the CPU before a branch is taken. Then if the branch is a misprediction, restore the CPU to correct state. Don't store to memory until it is determined this is the correctbranch.

CISC Processor: complex instruction set **CMP:** chip multiprocessor

coarse multi-threading: the thread being processed changes every few clock cycles **consistency:** order of access to differentaddresses
control hazard: branching and jumps cannot be executed until the destination address is known **CPI:**cycleperinstruction

CPU: central processing unit
Dark Silicon: the gap between how many transistors are on a chip and how many you can usesimultaneously. The simultaneous usage is determined by the power consumption of the chip.**datahazard:**theorderoftheprogramischangedwhichresultsindatacommandsbeingout of order, if the instructions are dependent - then there is a datahazard.

DDR SDRAM: double data rate synchronous dynamic RAM **dependency chain:** long series of dependent instructions in code

directory protocols: information about each block state in the caches is stored in a commondirectory.

DRAM: dynamic random access memory
DSM: distributed shared memory - all processors can access all memory locations **Enterprise class:** used for large scale systems that service enterprises

error: defect that results in failure
error forecasting: estimate presence, creation, and consequences of errors **error removal:** removing latent errors byverification

exclusionproperty:eachcachelevelwillnotcontainanydataheldbyalowerlevelcache**explicitILP:**co mpiler decides which instruction to execute inparallel

failure: the cause of an error
fault avoidance: prevent an occurrence of faults by construction

faulttolerance:preventfaultsfrombecomingfailuresthroughredundancy**faults:**actualbehavior deviates from specifiedbehavior

FIFO: first in first out
fine multi-threading: the thread being processed changes every cycle **FLOPS:** floating point operations per second

Flynn's Taxonomy: classifications of parallel computer architecture, SISD, SIMD, MISD, MIMD

FPR: floating point register **FSB:** front side bus

Geometric Mean: the nth root of the product of the numbers **global miss rate:** (the # of L2 misses)/(# of all memory misses) **GPR:** general purpose register

hit latency: time it takes to get data from cache. Includes the time to find the address in thecache and load it on the datalines

ilp: instruction levelprogramming
inclusion property: each level of cache will include all data from the lower level caches **IPC:** instructions percycle

IronLaw:executiontimeisthenumberofexecutedinstructionsN(writeNinintheExeTimefor Single-Cycle), times the CPI (write x1), times the clock cycle time (write 2ns) so we get N2*ns (write =N*2ns) forsingle-cycle.

Iron Law: instructions per program depends on source code, compiler technology, and ISA. CPI depends upon the ISA and the micro architecture. Time per cycle depends upon the micro architecture and the basetechnology.

iron law of computer performance: relates cycles per instruction, frequency and number of instructions to computerperformance
ISA: instruction set architecture
Itanium architecture: an explicit ILParchitecture,
sixinstructionscanbeexecutedperclockcycle

ItaniumProcessor: Intelfamilyof64-bitprocessorsthatusestheItaniumarchitecture**LFU:**least frequentlyused

ll and sc: load link and store conditional, a method using two instructions ll and sc for ensuringsynchronization.

local miss rate: # of L2 misses/ # of L1 misses
locality principle: things that will happen soon are likely to be similar to things that just happened.

loop interchange: used for nested loops. Interchange the order of the iterations of the loop, to make the accesses of the indexes closer to what is actually the layout inmemory

LRU: least recently used **LSQ:** load store queue

MCB: memory conflict buffer - "Dynamic Memory Disambiguation Using the Memory Conflict Buffer", see also "MemoryDisambiguation"

MEOSI Protocol: modified-exclusive-owner-shared-invalid protocol, the states of any cachedblock.

MESI Protocol: modified-exclusive-shared-invalid protocol, the states of any cached block. **Message Passing:** a processor can only access its local memory. To access other memorylocationsismustsendrequest/receivemessagesfordataatothermemorylocations.**meta-predictor:** a predictor that chooses the best branch predictor for each branch. **MIMD:** multiple instruction stream, multiple datastreams

MISD: multiple instruction streams, single data stream
misslatency:timeittakestogetdatafrommainmemory.Thisincludesthetimeittakestocheck that it is not in the cache and then to determine who owns the data, and then send it to theCPU.

mobo: mother board
Moore'sLaw: GordonE.Mooreobservedthenumberoftransistorsonanintegratedcircuitboard doubles every twoyears.

MP: multiprocessing
MPKI: Misses per Kilo Instruction

MSIProtocol:modified-shared-invalidprotocol,thestatesofanycachedblock.**MTPI:**message transfer partinterface

MTTF: mean time to failure **MTTR:** mean time to repair

multi-levelcaches:cacheswithtwoormorelevels,eachlevellargerandslowerthantheprevious level

mutex variable: mutually exclusive (mutex), a low level synchronization mechanism. A thread acquires the variable, then releases it upon completion of the task. During this period no other thread can acquire themutex.

NMRU: not most recently used

non-blocking caches: if there is a miss, the cache services the next request while waiting formemory

NUMA: non-uniform memory access, also called a distributed shared memory **OOO:** out of order
OS: operating system
PAPT: physically addressed, physically tagged cache - the cache stores the data based on its physcialaddress

PC: program counter
PCI: peripheral component interconnect
Pentium Processor: x86 super scalar processor from Intel
physical registers: registers, FP and GP that are not visible to the programmer **pipeline burst cache:**

pipelined cache: a pipelined burst cache uses 3 clock cycles to transfer the first data set from acacheblock,then1clockcycletotransfereachoftherest.Thepipelineandthe'burst'.(3-1-1-1) **PIPT:** physically indexed, physically tagged cache.

Power: Power = 1/2C *V^2 * f* Alpha
Power Architecture: performance optimization with enhanced RISC

Power vs Performance Equation:
pre-fetch buffer: when getting data from memory, get all the data in the row and store it in abuffer.

pre-fetching cache: instructions are fetched from memory before they are needed by the cpu "**PrescottProcessor:**BasedontheNetburstarchitecture.Ithasa31stagepipelineinthecore.The high penatly paid for mispredictions is supposedly offset with a Rapid Execution Engine. It also has a trace execution cache, this stores decoded instructions and then reuses them instead of fetching and decodingagain.

PRF: physical register file
pseudo associative cache: an address is first searched in 1/2 of the cache. If it is not there, thenit is searched in the other half of thecache.

RAID: redundant array of independent disks
RAID0:stripsofdataarestoredondisks-alternatingbetweendisks.Eachdisksuppliesaportion of the data, which usually improvesperformance.

RAID1:thedataisreplicatedonanotherdisk.Eachdiskcontainsthedata.Whicheverdiskisfree respondstothereadrequest.Thewriterequestiswrittentoonediskandthenmirroredtotheother disk(s).

RAID 2 and RAID 3: the data is striped on disks and Hamming codes or parity bits are used for error detection. RAID 2 and RAID 3 are not used in any currentapplication

RAID 4: Data is striped in large blocks onto disks with a dedicated parity disk. It is used by the NetAppcompany.

RAID5: Dataisstripedinlargeblocksontodisks,butthereisnodedicatedparitydisk.Theparity for each block is stored on one of the datablocks.

RAR: read after read **RAS:** return address stack **RAT:**register alias table
RAT: *(another RAT in multiprocessing) register allocation table **RAW:** read after write

RDRAM: direct random access memory
relaxed consistency: some instructions can be performed ooo and still maintain consistency**reliability:** measure of continuous service accomplishment

reservation stations: function unit buffers
RETO: return from interrupt
RF: register file
RISC Processor: reduced instruction set - simple instructions of the same size. Instructions are executed in one clockcycle

ROB: re-order buffer **RS:** reservation station

RWX: read - write- execute permissions on files
SHARC processor: floating point processors designed for DSP applications **SIMD:** singe instruction stream, multiple data streams

simultaneous multi-threading: instructions from different threads are processed, even in the samecycle

SISD: single instruction stream , single data stream**SMP:** symmetric multiprocessing

SMT: simultaneous multi threading
snooping protocols: A broadcast network - caches for each processor watch the bus for addresses in their cache.

SPARC processor: Scalable Processor Architecture -a RISC instruction set processor
spatial locality: if we access a memory location, nearby memory locations have a tendency to beaccessedsoon.

Speedup:howmuchfasteramodifiedsystemiscomparedtotheunmodifiedsystem.**SPR:**special purpose registers - such as program counter, or statusregister

SRAM: static random access memory
structural hazard: the pipeline contains two instructions attempting to access the same resource.

superscalararchitecture:theprocessormanagesinstructiondependenciesatrun-time.Executes more than one instruction per clock cycle usingpipelines.

synchronization: "a system is sequentially consistent if the result of any execution is the sameas if the operations of all the processors were executed in some sequential order, and the operations for each individual processor appear in the order specified by the program." Quote by Leslie Lamport

Synchronous Message Passing: a processor requests data then waits until the data is received beforecontinuing.

tag:thepartofthedataaddressthatisusedtofindthedatainthecache.Thisportionoftheaddress is unique so that it can be distinguished from other lines in thecache.

temporal locality: if a program accesses a memory location, it tends to access the same location again verysoon.

TLB: translation look aside buffer - a cache of translated virtual memory to physical memoryaddresses. TLB misses are very time consuming

Tomasulo's Algorithm: achieve high performance without using special compilers by using dynamicscheduling

tournament predictor: a meta-predictor
trace caches: sets of instructions are stored in a separate cache. These are instructions that have been decoded and executed. If there is a branch in the set, only the taken branch instructions are kept. If there is a misprediction the tracestops.
trace scheduling: rearranging instructions for faster execution, the common cases are scheduled
tree, tournament, dissemination barriers: types of structures forbarriers

UMA: uniform memory access - all memory locations have similar latencie.