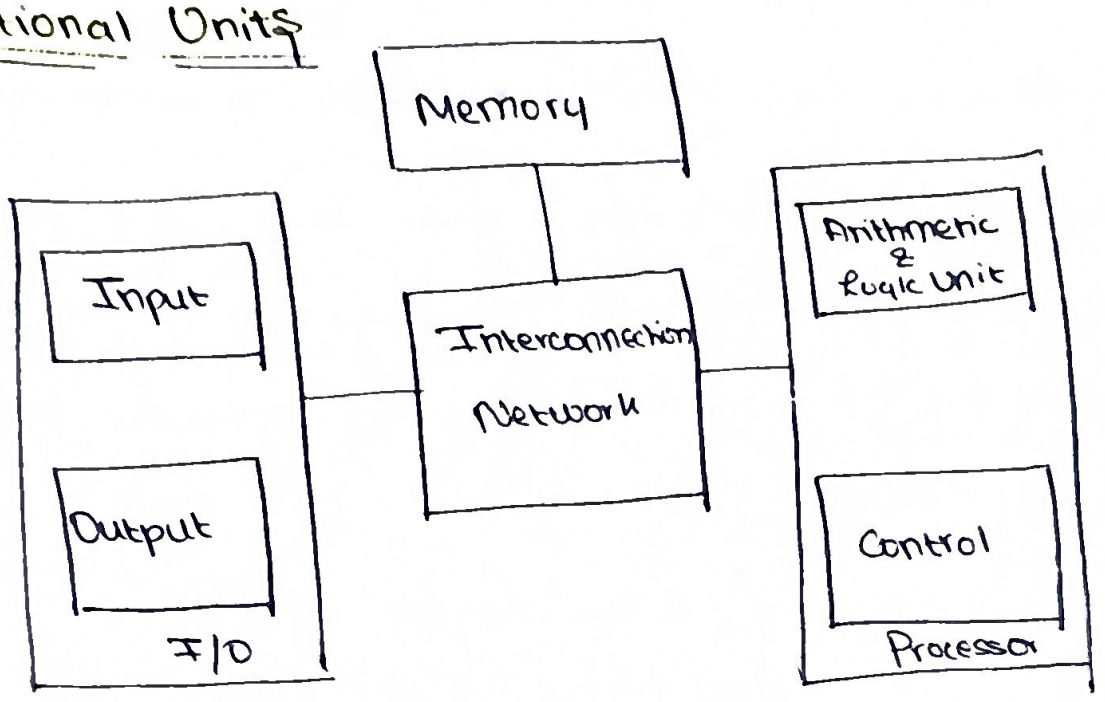
Computer Organization and Architecture

Unit I

Basic Structure of a Computer System

* Functional Units



→ A computer consists of 5 functionally independent main parts: input, memory, arithmetic & logic unit, output & control units

→ The input unit accepts coded information from human operators using devices such as keyboards, or from other computers.

→ The information received is stored in the computer's memory, to be processed immediately, or for later use, by the arithmetic & logic unit.

→ The processing steps are specified by a program that is also stored in the memory.

→ The results are sent back out via the output unit

→ All of those actions are controlled by the control unit

→ An interconnection network provides a means for the functional units to exchange information and coordinate their actions.

* Information handled by a computer

A. Instructions

→ are explicit commands that govern the transfer of information within a computer as well as between computers and I/O devices

→ specifies the arithmetic and logic operations to be performed

→ A list of instructions that performs a task is called a program. The program is stored in the memory, the processor fetches the instructions that make up the program from the memory and perform the desired operations.

B. Data

→ refers to numbers and encoded characters that are used as operands by the instructions.

→ Data can also refer to entire programs, that have to be processed by another program. (eg. compiling a high-level lang. source program into a list of machine instructions, called the object program).

→ Data has to be encoded in a binary format, 0s and 1s.

* Explanations about each Functional Unit

① Input Unit

→ computers accept coded information through input units, which read the data.

→ Any input is automatically translated into its corresponding binary code and transmitted to the processor.

→ eg. keyboard, touchpad, mouse, microphones, camera.

② Output Unit

→ sends processed results to the outside world

→ eg. display, printer, speakers

Note that some units, such as graphic displays provide both an output function and an input fn – I/O unit considered as one entity

③ Memory Unit

→ to store programs and data.

There are 2 classes of storage, primary & secondary memory

↓ fast storage, can be accessed easily. Programs must be stored in the primary memory when being executed

↓ used when there is a large no. of programs and info that is accessed infrequently.

Basis for Comparison	Cache Memory	Primary Memory	Secondary Memory
Basic	Cache + processor in same chip	directly accessible by praessor	not directly accessible by processor
Altered Name	NA	Main memory	Auxilliary memory
Data	Frequently used data & instructions	Instructions or data to be currently executed are copied to the main memory	Data to be permanently stored in secondary memory
Volatility	Volatile	Volatile	Non-Volatile.
Made of	Transistors	Semiconductors	Magnetic & optical material
Access Speed	Fastest	Faster than 2o	Slowest
Means of Access	data bus	data bus	I/O channels
Size	Max in MB	in GB	in TB
Cost	Most expensive	expensive	cheap
Location	part of internal memory	part of internal memory	external memory
Examples	NIL	RAM, ROM	Harddisk, pendrive

④ Arithmetic and Logic Unit

→ Coordinates the sequence of data movements into, out of & between processor subunits

→ Interprets instructions

→ Controls data flow inside processor

→ Handles task like fetching, decoding, execution handling & storing results

⑤ Control Unit

→ coordination of operations performed by the memory, ALU, I/O units

→ sends control signals to other units and senses their states

→ CU generates timing signals that govern the I/O transfers.

→ Much of the CU's circuitry is distributed throughout the machine

* CPU (Processor)

→ the brain of the machine.

→ responsible for carrying out all functional tasks

→ contains ALU, CU, registers

ALU → performs arithmetic & logical operations

CU → provides control signals in accordance to timing signals

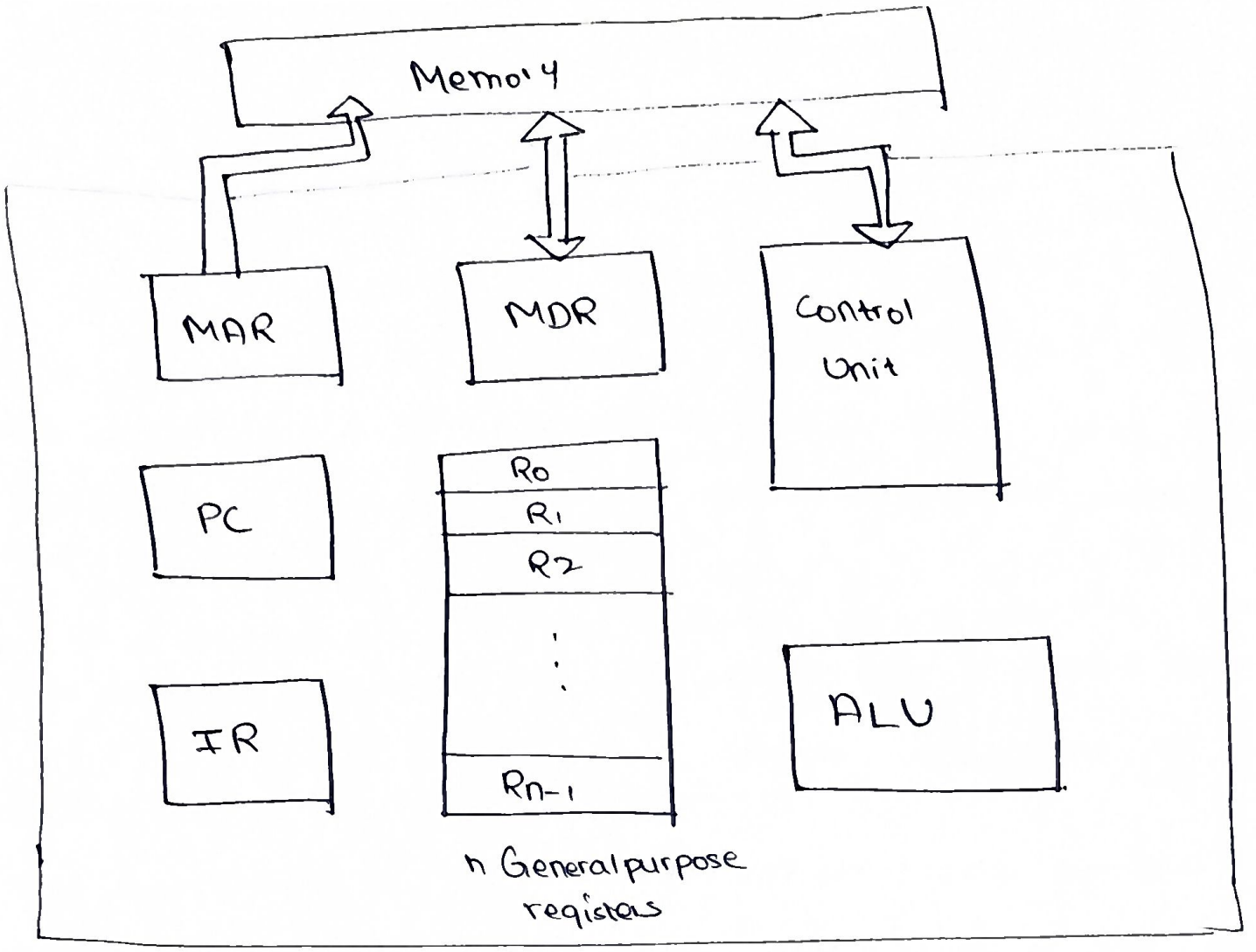
registers → store data & results & speeds up operation

* Basic Operational Concepts

→ Along with the memory, the processor contains the ALU, CU, and 2 types of registers

(i) General Purpose Registers - there are n general purpose registers

(ii) Special purpose registers

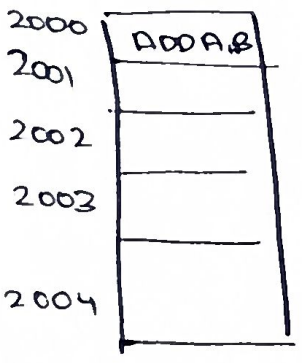


① Program Counter

→ Keeps track of the instructions of the program to be executed

→ It always contains the address of the next instruction to be fetched and executed.

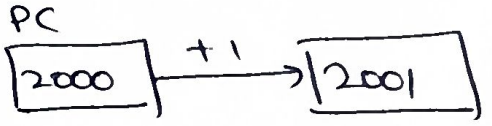
eg. If there are 5 instructions to be executed



PC | 2000

→ Initially, the PC has the address of the first instruction to be executed.

→ During the execution of the current instruction, the PC is incremented to the next instruction's address.



* Special Purpose Registers

A. IR - Instruction Registers

→ holds the instruction that is currently being executed

→ The content of the IR is sent to the CU, which generates timing signals, that are involved in the execution of the instruction.

2. MAR - Memory Address Register

The memory address reg contains the address of the location to be accessed in the memory.

3. MDR - Memory Data Register

→ Contains data to be read from memory from a particular location or write a data into a particular memory location in the main memory.

eg. If one wants to access the no. 30 from the memory unit, say its address is 2004.

→ This address is held by the MAR

→ later, the corresponding value (30) is fetched from the address and stored in the MDR.

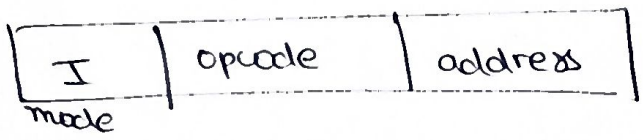
* ALU and CU

ALU → can perform arithmetic operations like +, −, *, ÷ as well as logical operations like AND, OR, NOT, EX-OR

CU → issues control signals through timing signals
The FU are controlled by the CU

* Memory

→ Both the programs and data are stored in the memory
→ stored in a 16 bit instruction format → has the address, mode & opcode fields



Mode → denoted by I

I=0 ⇒ direct addressing, directly get operand from the address specified
I=1 ⇒ indirect addressing, access the indirect address in the memory to get the memory of the operand

Opcode - specifies what operation is to be performed

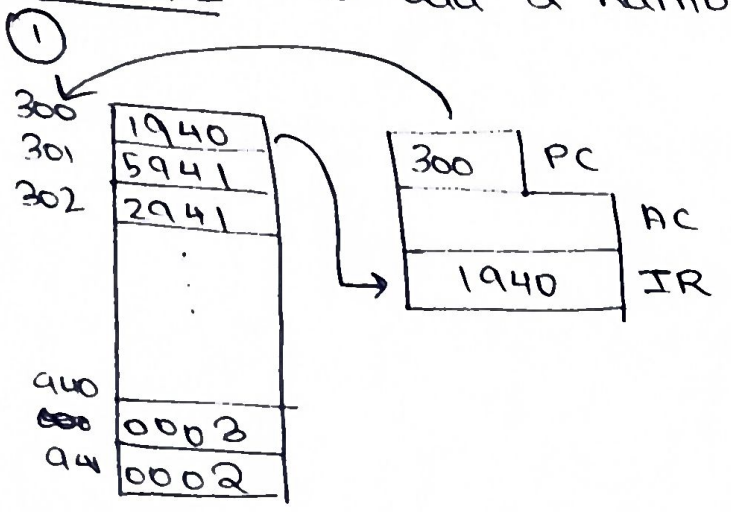
* Sequence of Operations

1. Initially, the PC points to the first instruction. (say the 1st address is 2000).
2. The content of the PC is transferred to the MAR.
3. The MAR now contains the address, and a read operation can be performed on it
4. The data/ corresponding to the address in the MAR is sent to the MDR. inst
5. The content of the MDR is transferred to the IR. The IR now has the instruction to be executed.
6. During the execution of the instruction, the PC is incremented.
7. The content of the IR is sent to the CU.
8. The CU can generate timing signals and send them to the functional units involved in the instruction.
9. In order to actually perform the operation, apply the DECODE operation to the OPCODE. (say, its an ADD operation)
10. The operands required for the ADD operation are stored either in the memory unit or in the processor registers.
11. Suppose the 2 operands are stored in the memory at particular addresses specified by the MAR.
12. The MAR now performs a read cycle in the memory unit.
13. The operands read are now sent to the MDR.
14. Now the MDR has both the operands, and the operation to be performed is also known.

15. The ALU performs the addition operation and the result is sent to the MDR.

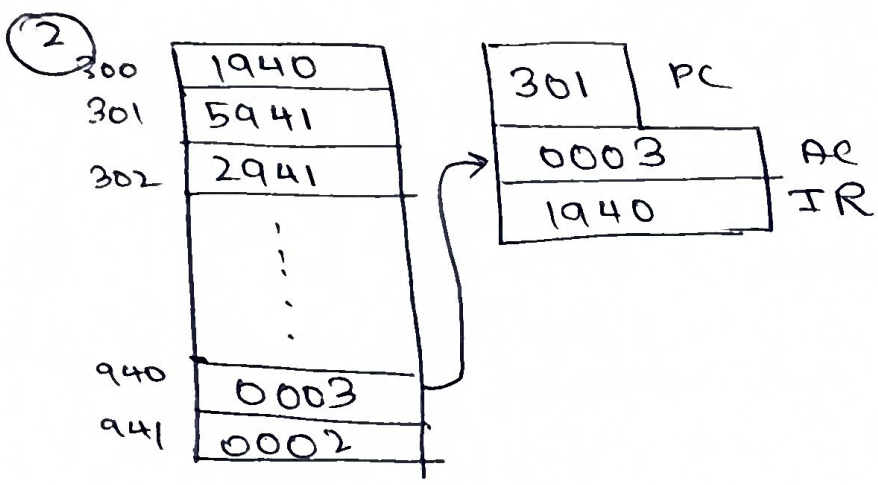
16. This can be stored in the MAR, at a specified address / can be sent to the o/p

Example: To add 2 numbers and store the result



PC holds address of 1st instruction

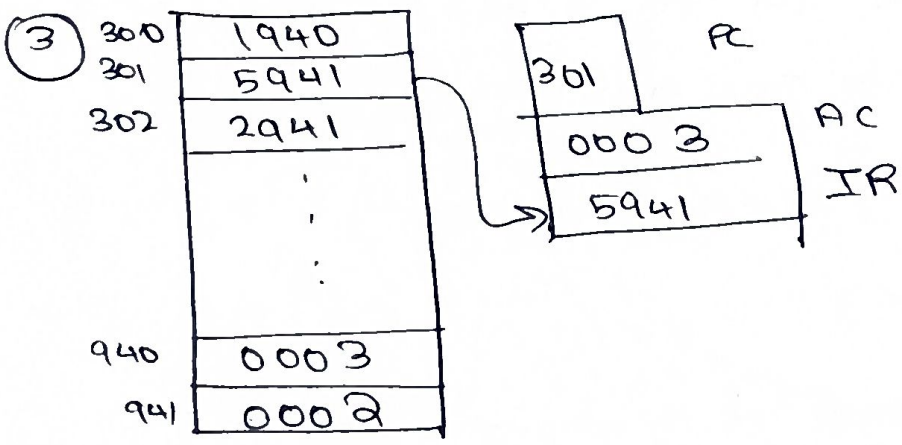
First instruction sent to IR



The MSB (1) specifies addition

Store value at given address (940) in the accumulator

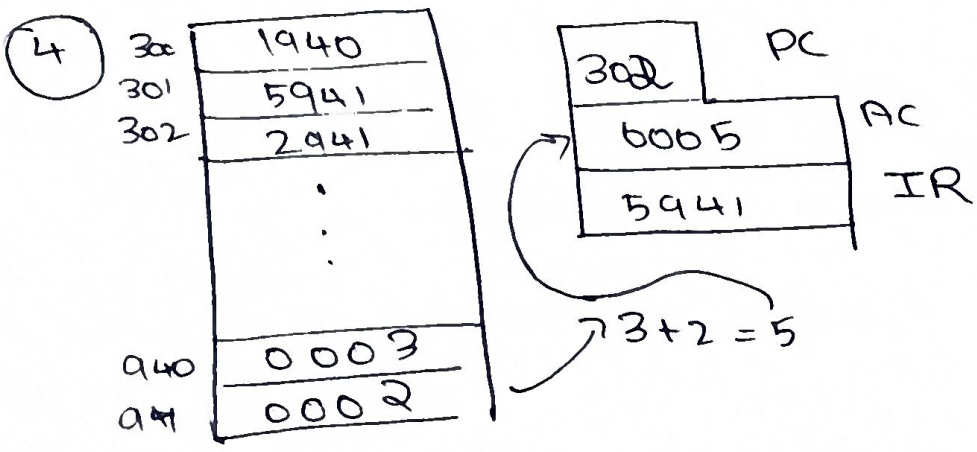
Increment PC



Read next instruction specified by address at PC (5941)

Store that instruction in the IR

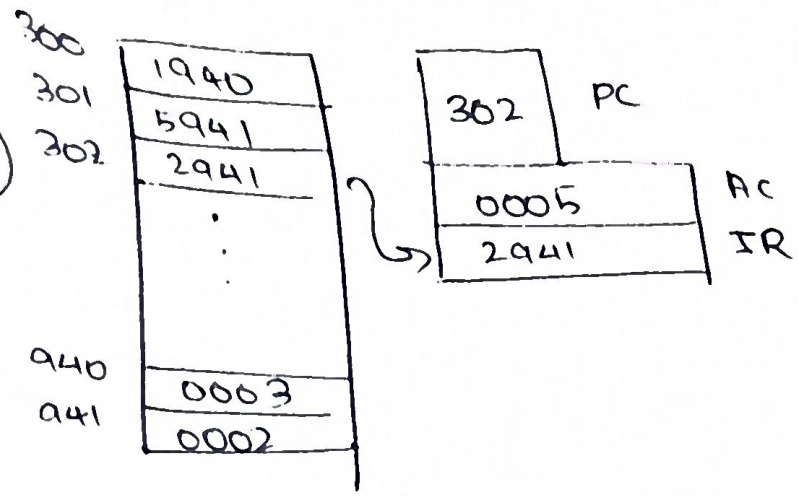
Increment PC



Add the 2 nos.

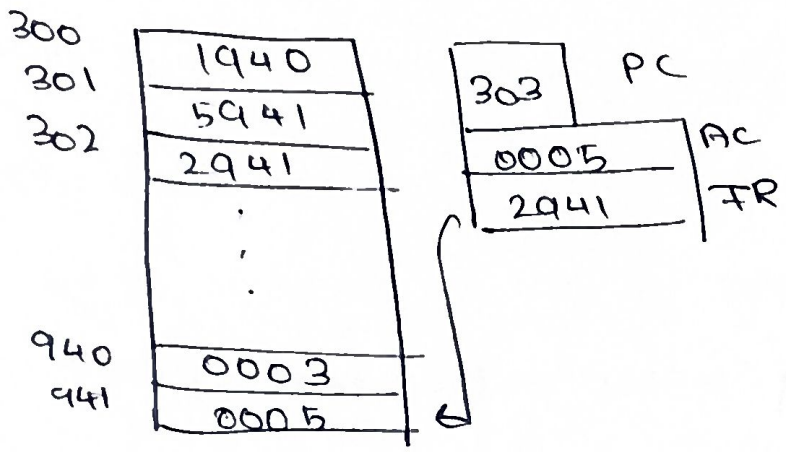
Store result back in accumulator

(5)



Read next instruction specified by address at PC (2941)

Store that instruction in the IR



Increment PC

Interpret instruction (store result back at address 941)

* Interrupts

→ Normal execution of programs may be interrupted if some device requires urgent servicing

→ To deal with the situation immediately, the normal execution of the current program must be interrupted.

Interrupt Procedure

→ The device raises an interrupt signal

→ The processor provides the requested service by executing an appropriate interrupt - service routine.

→ The state of the processor is first saved before servicing the interrupt.

→ When the interrupt service routine is completed, the state of the processor is restored, so that the interrupted program may continue.

* Bus Structures

→ For a computer to perform different operations, the functional units need to communicate with each other.

→ The FU are connected by a group of parallel wires called a bus.

→ Each wire in a bus can transfer one bit of information.

→ The no. of parallel wires in a bus is equal to the word length of a computer.

Drawbacks of a single bus structure

→ The devices connected to a bus vary widely in the speed of their operation

printers & keyboard are slow

optical disks are fast

memory & processor units are the fastest

→ An efficient transfer mechanism is needed to cope w/ this problem

Solutions:
- use buffer registers to hold info during transfer
- use a 2 bus structure & an additional transfer mechanism
- a high-performance bus, a low-performance & a bridge for transferring data between the 2 buses is an option

The ARMA bus belongs to this structure

↓

Advanced Microcontroller Bus Architecture

* Performance

→ how quickly a computer can execute programs

→ For best performance, it is necessary to design the compiler, the machine instruction set & hardware in a coordinated way.

* Response Time : The time it takes for a response for a single program – how long it takes from start to finish

* Latency : Amount of time it takes to complete one unit of work

$$\text{Latency} = \frac{\text{Time}}{\text{1 work unit}}$$

* Throughput : no. of work units that can be completed in a given amount of time.

i.e the number of bits or bytes that are transmitted over a network in a given amount of time.

$$\text{Throughput} = \frac{\text{work units}}{\text{Time}}$$

* Parallelism and Pipelining

→ performance can be increased by performing a number of operations in parallel.

→ Instruction-level parallelism involves the overlap of the execution of instruction.

→ This reduces total execution time.

→ For eg. the next instruction could be fetched from the memory at the same time that an arithmetic operation is being performed on the operands of the current instruction.

→ This form of parallelism is called pipelining.

* Speed Up : Machine A is said to be n times faster than B if

$$S = \frac{\text{time}(B)}{\text{time}(A)} = n$$

if $n > 1$ ⇒ speedup, $n < 1$ ⇒ slowdown

% speedup $= (S-1) * 100\%$

* Iron Law of Performance

$$\text{Processor Performance} = \frac{\text{Time}}{\text{Program}}$$

$$= \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{cycle}}$$

$$T = I \times CPI \times C$$

(i) Instructions / Program - determined by instructions executed, not static code size (determined by algo, compiler.)

(ii) Cycles / Instruction: determined by ISA (Instruction Set Architecture) and processor organization

overlap of instruction reduces this term.

(iii) Time / Cycle : determined by technology & microarchitecture

Example 1: A program is running on a specific machine with the following parameters:

Total executed instruction count = 10,000,000 instruction

Average CPI for the program = 2.5 cycles / instruction

CPU clock rate = 200 MHz

What is the execution time for this program?

Solution $f = 200\ MHz$

$$\text{Time / cycle} = 1/200 \times 10^6 = \frac{1}{2} \times 10^{-8}$$

$$C = 0.5 \times 10^{-8} = 5 \times 10^{-9}\ s$$

$$\text{Execution Time} = I \times CPI \times C$$

$$= 10,000,000 \times 2.5 \times 5 \times 10^{-9}$$

$$= 1 \times 10^7 \times 2.5 \times 5 \times 10^{-9}$$

$$= 12.5 \times 10^{-2} = \boxed{0.125 \text{ seconds}}$$

Example 2: A program is running on a specific machine (CPU) with the following parameters:

Total executed instruction count, I = 10,000,000 instructions

Average CPI for the program = 2.5 cycles per instruction

CPU clock rate = 200 MHz

Another compiler is used with a new executed instruction count

I = 9,500,000

New CPI = 3.0, and the new clock rate = 300 MHz

Compute the speedup?

Solution

$$\text{Speedup} = \frac{\text{Old Execution Time}}{\text{New Execution Time}} = \frac{I_{old} \times CPI_{old} \times \text{Clock cycle}_{(old)}}{I_{new} \times CPI_{new} \times \text{Clock cycle}_{(new)}}$$

$$= \frac{10^7 \times 2.5 \times \frac{1}{200}}{9.5 \times 10^6 \times 3 \times \frac{1}{300}}$$

$$= \frac{10^7 \times 1.25}{10^6 \times 9.5}$$

$$= \frac{1.25}{9.5} \times 10 = 1.3157$$

$\%$ speedup $= (1.3157 - 1) \times 100\%$

$= 31.57\%$

* Instruction Types and CPI

Given a program with n types or classes of instructions executed on a given CPU, with the following characteristics:

C_i = count of instructions of type i executed

CPI_i = cycles per instruction for type i

$$CPI = \text{CPU clock cycles} / \text{Instruction count } I$$

$$\text{CPU clock cycles} = \sum_{i=1}^{n} CPI_i \times C_i$$

$$I = \sum C_i$$

Example 3: An instruction set has 3 instruction classes

Instruction class	CPI
A	1
B	2
C	3

Two code sequences have the following instruction counts

Code sequence	Instruction counts for instruction class		
	A	B	C
1	2	1	2
2	4	1	1

Find the CPI for each of the code sequences

Solution For Code Sequence 1:

CPU clock cycles = $(1\times2) + (1\times2) + (2\times3) = 10$

$I = 5$

$CPI = CPU / I = 2$ //

For Code Sequence 2 CPU clock cycle = $(4\times1) + (1\times2) + (1\times3) = 9$

$I = 6$

$CPI = CPU / I = 9/6 = 1.5$ //

* Instruction Frequency and CPI

Given a program with n types or classes of instructions with the following characteristics.

C_i = count of instructions of type i executed

CPI_i = avg. cycles per instruction of type i

F_i = frequency or fraction of instruction of type i executed

$= C_i / I$

Avg. or effective CPI

$$CPI = \sum_{i=1}^{n} (CPI_i \times F_i)$$

Fraction of total execution time for instructions of type i

$$= \frac{CPI_i \times F_i}{CPI}$$

Example 4 : Consider the following type of instructions & their percentage frequency. Compute the average or effective CPI and the fraction of total execution time for each type of instruction

Instruction Type	F_i	CPI_i
ALU	50%	1
Load	20%	5
Store	10%	3
Branch	20%	2

Solution: $CPI = \sum_{i=1}^{n} (CPI \times F_i)$

$= (0.5 \times 1) + (0.2 \times 5) + (0.1 \times 3) + (0.2 \times 2)$

$= 0.5 + 1 + 0.3 + 0.4 = 2.2$

Fraction of time for each

(i) ALU = $0.5 / 2.2 = 0.22$

(ii) Load = $1 / 2.2 = 0.45$

(iii) Store $0.3 / 2.2 = 0.136$

(iv) Branch = $0.4 / 2.2 = 0.18$

Example 5 : Computer A has a 2GHz clock, 10s CPU time. Computer B must be designed such that it has a CPU time of 6s. It must have 1.2 times the clock cycles of Computer A. How fast must Computer B's clock be?

Solution

Computer A

Clock rate = 2GHz

CPU Time = 10s

$$\text{CPU Time} = \frac{\text{clock cycles}}{\text{clock rate}}$$

clock cycles = $10 \times 2 \times 10^9 = 20 \times 10^9$ cycles

Computer B:

clock cycles = 1.2 x clock cycles (A)

$= 1.2 \times 20 \times 10^9$

$= 24 \times 10^9$

CPU Time = 6s

$$\text{Clock rate} = \frac{\text{clock cycles}}{\text{CPU time}} = \frac{24 \times 10^9}{6} = 4 \times 10^9 = 4\text{GHz}$$

Example 6 : Computer A has a cycle time of 250ps & CPI of 2.0. Computer B has a cycle time of 500ps and a CPI of 1.2. Both have the same ISA.

Which is faster and by how much

Execution Time$_A$ = I x CPI x C = I x 2.0 x 250ps = 500I

Execution Time$_B$ = I x CPI x C = I x 1.2 x 500ps = 600I

A is faster by $\frac{\text{Execution Time B}}{\text{Execution Time A}} = \frac{600I}{500I} = 1.2$ times

* Performance Evaluation in Real World Applications

→ The System Performance Evaluation Corporation (SPEC) selects and publishes representative applications for different domains.

→ These programs are used to measure performance, that are supposedly typical of the actual workload.

→ SPEC rating $= \frac{\text{running time on reference computer}}{\text{running time on the computer under test}}$

→ The SPEC CPU2006 uses the following criteria

(i) find elapsed time to execute a selection of programs

(ii) normalize relative to reference machine

(iii) summarize as the geometric mean of performance ratios

$$\text{Spec rating} = \left(\prod_{i=1}^{n} \text{SPEC } i \right)^{1/n}$$

n = no. of programs in the suite

* **MIPs Rating** : The MIPs rating is a measure of how many millions of instructed are executed per second

$$\text{MIPs Rating} = \frac{\text{Instruction Count}}{\text{Execution Time} \times 10^6}$$

Pitfalls - doesn't account for → differences in ISA between computer
→ differences in complexity between instructions

Example 7 : Let $CPI_A = 2.0$ & $CPI_B = 1.2$. For equal performance, if clock (B) = 2ns, what does clock (A) need to be?

Solution : Execution Time $_A$ = Execution Time $_B$

$$I \times CPI_A \times C_A = I \times CPI_B \times C_B$$

~~$2 \times 1.8 = 1.2 \times C_B$~~

~~C_B~~ : $2 \times C_A = 1.2 \times 2$

$$C_A = 1.2 \text{ ns}$$

Examples : Suppose there are 2 computers A and B.

Computer A : clock cycle = 1ns

performs 2 instructions / second

Computer B : clock cycle = 600 ps

performs 1.25 inst / sec

Assuming that a program requires the execution of the same no. of instructions in both computers. Which computer is faster?

Solution : Execution Time = I x CPI x C

Computer A = ~~I x C~~

$$\frac{2 \text{ inst}}{1 \text{ cycle}} \times \frac{1 \text{ cycle}}{10^{-9}}$$

$= 2 \times 10^9$ inst / second

Computer B = $\frac{1.25 \text{ inst}}{1 \text{ cycle}} \quad \frac{\times \text{ 1 cycle}}{600 \times 10^{-12} \text{ sec}}$

$= 2.08 \times 10^9$ inst / sec

Computer B is faster

Example 9 : Assume that a new webserver is 10 x faster than the previous web-server. I/O performance is not improved compared to the old machine. The web-server spends 40% of its time in computation & 60% in I/O. How much faster is the new machine overall?

$$\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{enh}) + \frac{\text{Fraction enhanced}}{\text{Speedup enhanced}}}$$

$\text{Fraction}_{enh} = 40\%$

$\text{Speedup}_{enh} = 10$

$$= \frac{1}{(1-0.4) + \frac{0.4}{10}}$$

$$= \frac{1}{0.6 + 0.04} \quad = \frac{1}{0.64} = 1.56$$

Example 10

* Instructions

* Big and Little Endian Notation

big endian - lower byte addresses become the MSB

little endian - lower byte addresses used for LSB

* Instructions - The tasks carried out by a computer consist of a sequence of small steps. These steps can do operations like adding numbers, testing for a particular condition, I/O operations

→ A computer must have instructions capable of performing 4 types of operations.

(i) Data transfer between the memory and processor registers

(ii) Arithmetic and logical operations

(iii) Program sequencing & control

(iv) I/O transfers

* Register Transfer Notation

→ To describe the transfer of information, the contents of any location are denoted by placing square brackets around its name.

Ex1

⇒ $R_2 \leftarrow [LOC]$

means that the content of memory location LOC are transferred to the processor register R_2

Example2 Add the contents of registers R_2 and R_3, and place their sum in register R_4.

$$R_4 \leftarrow [R_2] + [R_3]$$

→ This is called RTN, where the RHS is always a value and the LHS is the destination location.

* Assembly - Language Notation

→ notation used to represent machine instructions & programs.

Format

< Instruction > < destination > < Input regs . . . . >

eg. LOAD R_2, Loc ⇒ copy the contents of Loc into R_2

eg. ADD R_4, R_2, R_3 ⇒ add R_2 and R_3 and store the result in R_4

(The destination register gets overwritten)

* Instruction Formats

Format	Assembly	RTN
3 address instruction	ADD R_1, R_2, R_3	$R_1 \leftarrow [R_2] + [R_3]$
2 address instruction	ADD R_1, R_2	$R_1 \leftarrow R_1 + R_2$
1 address instruction	ADD M	$AC \leftarrow AC + M[AR]$
0 address instruction	ADD	$TOS \leftarrow TOS + (TOS-1)$

Example Express (A+B) * (C+D) in 3, 2, 1 and 0 address instruction formats.

(1) 3 address

ADD R_1, A, B ; $R_1 \leftarrow M[A] + M[B]$

ADD R_2, C, D ; $R_2 \leftarrow M[C] + M[D]$

MUL X, R_1, R_2 ; $M[X] \leftarrow R_1 * R_2$

2 address format

1. MOV R1, A — R1 ← M[A]
   ADD R1, B — R1 ← M[B] + R1
   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

(See ppt for the other 2 - not sure if in syllabus)

* RISC and CISC Instruction Sets

RISC pipelining

RISC

→ Each instruction fits into a single word

→ A LOAD/STORE architecture is used

- memory operands are accessed using only LOAD and STORE instructions

- All operands must be in the processor registers, or one of the operands may be given explicitly w/ the instruction word.

→ effective use of pipelining, eg. ARM

CISC → makes use of much more complex instructions, which may span more than one word of memory

→ Because each CISC command must be translated by the processor into tens or even hundreds of lines of microcode, it tends to run slower than an equivalent series of simpler commands.

→ any instruction takes many cycles to execute

eg. Intel Pentium

* Instruction Set Architecture → refers to the actual programmer visible instruction set.

→ The ISA defines registers

defines data transfer modes between registers, memory and I/O

* The MIPS Instruction Set

MIPS - Microprocessor without Interlocked Apelino Stages

Features

→ 32 programmer visible registers

→ 32 floating registers

→ status register

→ program counter

→ Every assembly language instruction is preceded by $.

→ Special purpose registers lo and hi are used to store the result of multiplication & division, and are not directly addressable The contents are accessed w/ special instructions mfhi (move from Hi) and mflo (move from lo)

→ In MIPS, the stack grows from high memory to low memory

* MIPS Registers

Register No	Representation	Description
0	zero	the value 0
1	$at	assembler temporary - reserved by the assembler
2-3	$v0 - $v1	values - from expression evaluation & function results
4 -7	$a0 - $a3	arguments - 1st 4 parameters subroutine

Register No	Representation	Description
8-15	$t0 - $t7	temporary
16-23	$s0 - $s7	saved fallows
24-25	$t8 - $t9	temporary - used if 8-15 run out
26-27	$k0 - $k1	reserved for use by the interrupt / trap handler
28	$gp	global pointer
29	$sp	stack pointer - points to last location on stack
30	$s8 | $fp	saved value / frame pointer
31	$ra	return address

* 5 kinds of instructions

(i) arithmetic (iii) data transfer (v) unconditional jump

(ii) logical (iv) conditional branching

(1) Arithmetic Operations

Instruction	Example	Meaning	Comments
add	add $1, $2, $3	$1 = $2 + $3	
subtract	sub $1, $2, $3	$1 = $2 - $3	
addimmediate	addi $1, $2, 100	$1 = $2 + 100	Note that there is no subi
add unsigned	addu $1, $2, $3	$1 = $2 + $3	
subtract unsigned	subu $1, $2, $3	$1 = $2 - $3	
multiply	mult $2, $3	Hi, Lo = $2 * $3	
multiply unsigned	multu $2, $3	Hi, Lo = $2 * $3	
divide	div $2, $3	Lo = $2 ÷ $3 Hi = $2 mode $3	Lo = quotient Hi = remainder

Instruction	Example	Meaning	Comments
add imm. unsign.	addiu $1, $2, 100	$1 = $2 + 100	
divide unsigned	divu $2, $3	lo = $2 ÷ $3 Hi = $2 mod $3	
Move from Hi	mfhi $1	$1 = Hi	used to get copy of Hi & lo, note that it is stored in $at
Move from lo	mflo $1	$1 = lo	

Example1 Convert to MIPS instructions

$f = (g+h) - (i+j)$

f, g, h, i, j in $s0 - $s4

Ans:

```
add  $t0, $s1, $s2
add  $t1, $s3, $s4
sub  $s0, $t0, $t1
```

Example2 : Convert to MIPS instructions

$g = h + A[8]$

g in $s1, h in $s2, base address of A in $s3

Solution base address of A in $s3
requires an offset of 32 (8×4 = 32)

```
lw   $t0, 32($s3)
add  $s1, $s2, $t0
```

Example3 $A[12] = h + A[8]$

h is in $s2, base address of A in $s3

```
lw   $t0, 32($s3)
add  $t0, $s2, $t0
sw   48($s3), $t0
```

for sw, the destination comes second

* Registers vs. Memory

→ Registers are faster to access than memory.

→ Operating on data in memory requires loads and stores, and more instructions to be executed.

→ The compiler must use registers for variables as much as possible, only spill to memory for less frequently used variables.

Memory Spill in Compiler Design – If there are not enough variables to hold all the variables, some variables may be moved to and from the RAM.

→ Over the lifetime of a program, a variable can be both spilled and stored in registers – this variable is considered spill.

* MIPs Zero Register

→ MIPs register no. 0 is the constant 0

→ cannot be overwritten

→ useful for common operations like moving between registers

```
add $t2, $s1, zero
```

* Sign Extension

→ used to represent a no. using more bits, but preserve the numeric value

In the MIPs instruction set

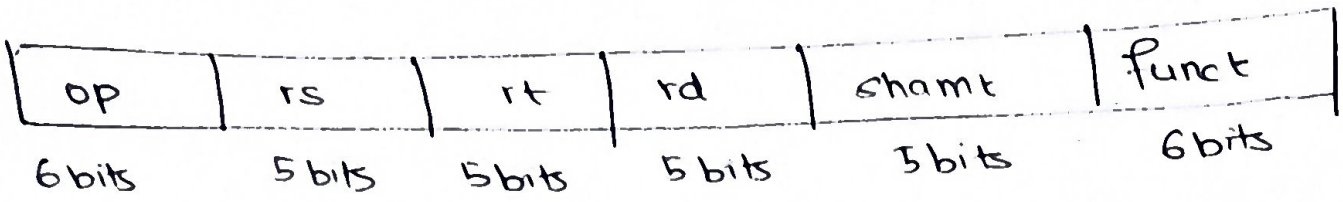
addi : extend immediate value

lb / lh : extend loaded byte / half word

beq, bne : extend displacements

for integers – extend signed bit leftwards

* MIPS R-Format Instructions



Instruction fields: (i) op: operation code (opcode)

(ii) rs : source register

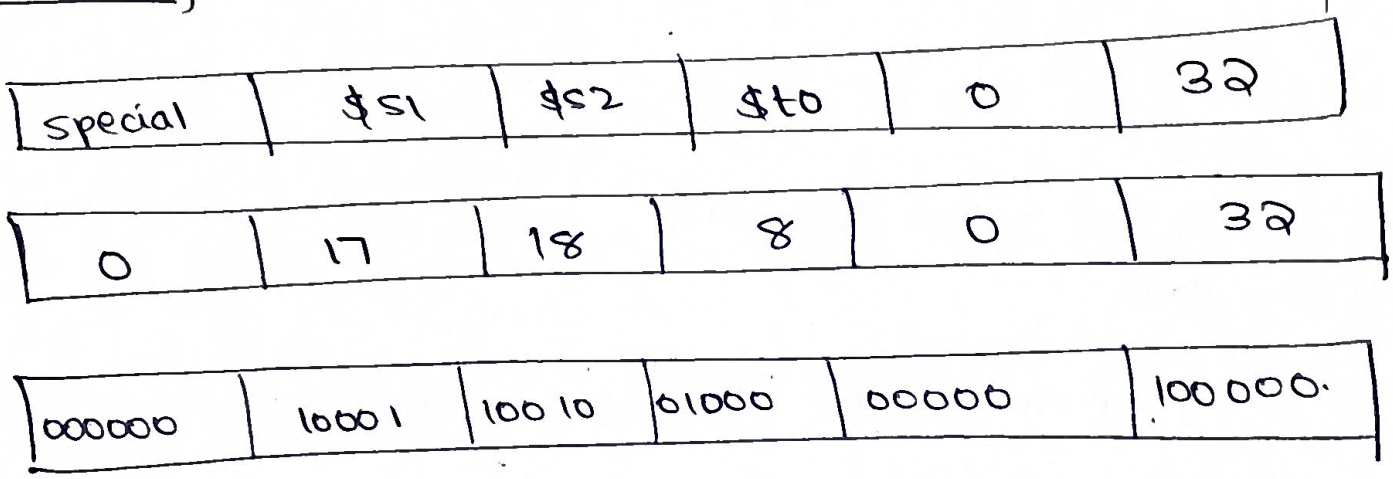
(iii) rt : second source register no.

(iv) rd : destination register

(v) shamt : shift amount

(vi) funct : function code (extends opcode)

Example1 add $t0, $s1, $s2



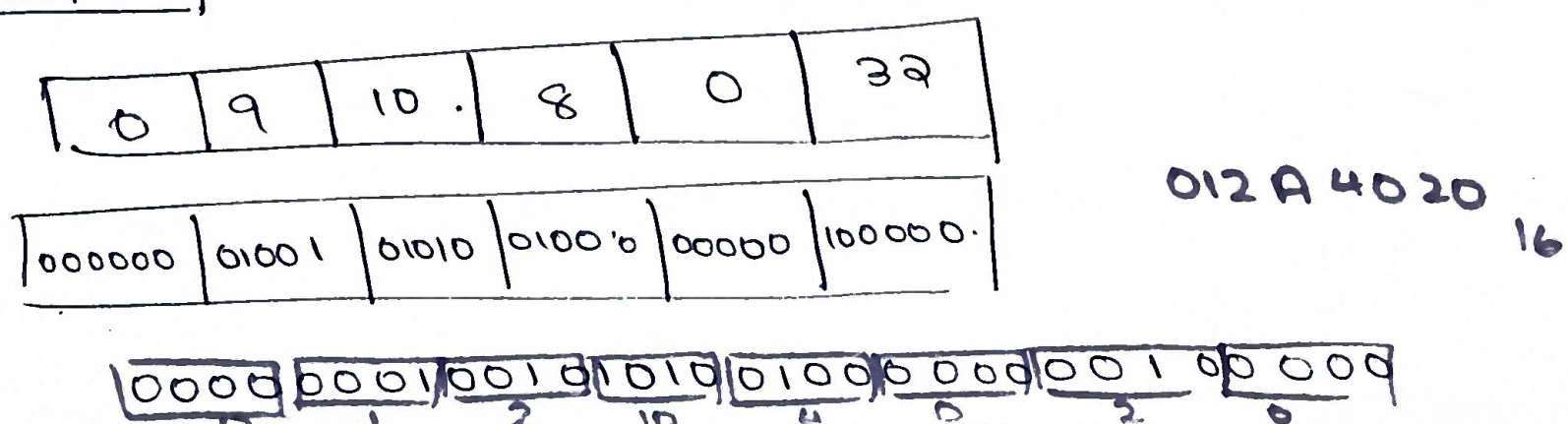
binary equivalent

000000 10001 10010 01000 00000 100000 $_2$

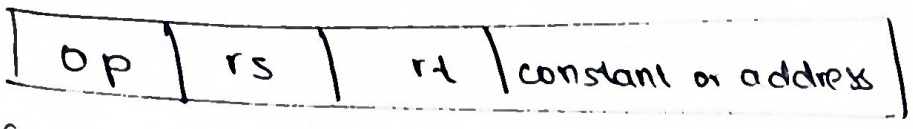
0 2 3 2 4 0 2 0

$= 02324020_{16}$

Example2 add $t8, $s9, $s10



* MIPS I-Format Instructions



eg. addi
slti
sltiu

→ can be used for immediate arithmetic & load & store instructions

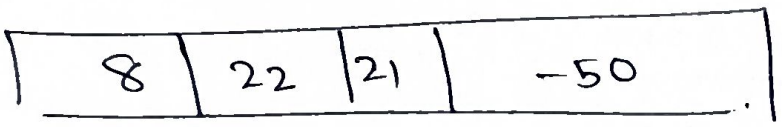
rt: destination or source register no. (target register)

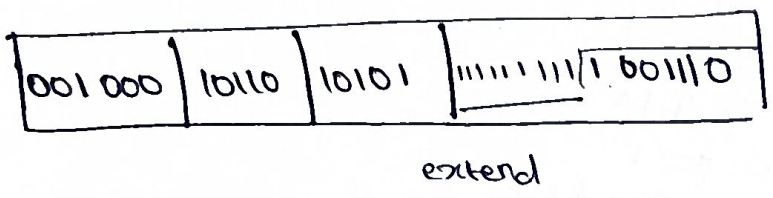
constant: -2^{15} to $+2^{15}-1$

rs: source register

Example: addi $21, $22, -50

addi opcode = 8





* Dealing with Large Immediates

→ There are 16 bits allotted in the I-format instructions for constants

→ The constants have a range of -2^{15} to $2^{15}-1$.

→ Sometimes one may want to use immediates > $\pm 2^{15}$ with addi, lw, sw, and slti

→ This is done by moving the upper half (bits 16-31) into one register, and making the rest of the bits 0, and taking the lower half and storing it in another reg, making the upper half 0.

→ Then an (or) operation is performed to get all 32 bits.

Example addiu $t0, $t0, 0xABABCDCD

⇒ lui $at, 0xABAB # upper 16

ori $at, $at, 0xCDCD

addu $t0, $t0, $at

* Conditional / Branching Instructions

Instruction	Example	Meaning	Comments
branch on equal	beq $1,$2,100	If $1==$2 goto PC+4+100	equality test
branch on not equal	bneq $1,$2,100	If $1!=$2 goto PC+4+100	not equal test
set on less than	slt $1,$2,$3	If $2<$3 $1=1 else $1=0	compare < than
set less than imm.	slti $1,$2,100	If $2<100, $1=1 else $1=0	compare < constant
set less than unsigned	sltu $1,$2,$3	If $2<$3, $1=1, else $1=0	
set less than imm. unsigned	sltu $1,$2,100	$2<100 $1=1 else $1=0	

→ For branching instructions, I format is used

opcode : beq = 4
bne = 5

→ Branches are typically used for loops

Example

```
loop: beq  $9, $0, End
      addu $8, $8, $10
      addiu $9, $9, -1
      j    loop
```

→ 1
→ 2
→ 3

Immediate = 3

PC relative addressing

use the Immediate field as the offset

If we don't take the branch

PC = PC + 4 = next inst

If we do take the branch

PC = (PC + 4) + (intermediate * 4)

immediate signifies the no. of instructions to jump forward or backward

opcode = 4
rs = 9
rt = 0

= (PC+4) + (3*4)

4	9	0	3

in binary

000100	01001	0000	0000000000011

If the destination is > 215 instructions away from the branch, use the instructions far and next

Example 2 The compiler puts f, g, h, i, j in $16, $17, $18, $19, $20

```
if (i == j)
    f = g + h
```

MIPS
```
bne $19, $20, endif
add $16, $17, $18
```

Example 3
```
if (i == j)
    f = g + h
else
    f = g - h
```

MIPS
```
bne $19, $20, else
add $16, $17, $18
J Exit
else: sub $16, $17, $18
exit:
```

* Assembler Pseudo Instructions / Assembler Directives

→ Certain statements are implemented unintuitively in MIPS.

→ MIPS has a set of pseudo-instructions to make programming easier. These get translated into actual instructions later

Ex1 `move dst, src` ⇒ (translated into) `addi dst, src, 0`

Ex2 `la dst, label` loads address of specified label into dst

Ex3 `li dst, imm` loads 32 bit immediate into dst

- .DATA – The following lines are constant or data values
- .WORD – reserve a 32 bit word in memory for a varicible
- .ASCII – place a string in memory using the ASCII character code
- .TEXT – The following lines are instructions

* Issues with pseudo instructions and the solution for it

Problem: When breaking up a pseudo instruction, the assembler may need to use an extra register

→ If it uses a regular register it'll overwrite whatever the program has put into it.

Solution: Reserve a register $1 or $at (assembler temporary) that the assembler will use to break up pseudo instructions

Since the assembler may use this at any time, it is not safe to code with it.

* MIPS labels

→ Instead of absolute memory addresses symbolic labels are used to indicate memory addresses in assembly language

eg. Variable X is stored in a location 123 in the memory. Use label X instead of 123 in programs

Ex1 N: .WORD 0

⇒ equivalent to variable declaration in a HLL

set up N as a label that points to a 32 bit value. Initial value 0

Ex2 LOOP: ADD $a0, $a0, $a1

Sets up LOOP as a label that points to the ADD instruction

* Additional Branching Examples

Example:

```
while (save[i] == k)
    i = i+1
```

i in $s3 , k in $s5 address of save in $s6

sl => *
sr => %
every left shift = * by 2

MIPS code

```
loop: sll $t1, $s3, 2
      add $t1, $t1, $s6
      lw  $t0, 0($t1)
      bne $t0, $s5, exit
      addi $s3, $s3, 1
      j loop
exit: ...
```

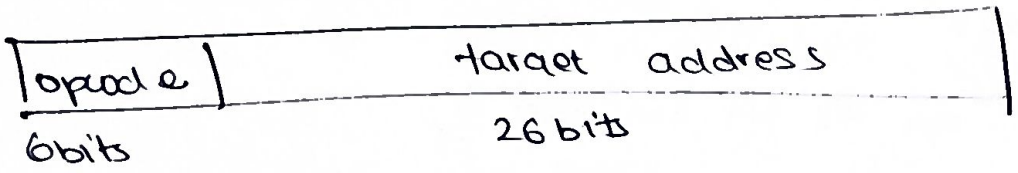
→ in $s3, t1 = s3 * 4

→ address of save[i]

* Jump Statements

Instruction	Example	Meaning	Comments
jump	j 10000	go to 10000	
jump register	jr $31	go to $31	used for subroutine returns
jump and link	jal 10000	$31 = PC+4 go to 10000	used for subroutine calls

→ For general jumps like j and jal, we may jump anywhere in the memory



→ Collapse all other fields ~~to~~ ~~no~~ after opcode to make room for large target address

→ can specify 2^{26} addresses

* Multiplication and Division

Multiplication

→ a type of pseudo instruction

Syntax: mul $rd , $rs, $rt

→ The pseudo instruction mul expands to mult / mflo

→ mult & div have nothing important in the rd field, since the destination registers are hi & lo.

→ Note that for m bits x n bits, the product would have m+n bits

→ In MIPs, registers are multiplied, so

-32 bit value x 32 bit value = 64 bit value

On multiplication, the upper half of the product is put in hi
lower half of the product is put in lo

Example a = b * c

let b = $s2 and a is $s0 and $s1
c = $s3

```
mult  $s2 , $s3
mfhi  $s0
mflo  $s1
```

Division

Syntax: div reg1 , reg2

divides 32 bit register 1 by 32 bit register 2

puts the remainder of division in hi, quotient in lo

eg. a = c/d

```
div  $s2 , $s3
mflo $s0
mfhi $s1
```

* Logical Operations

Instructions	Example	Meaning
and	and $1, $2, $3	$1 = $2 & $3
or	or $1, $2, $3	$1 = $2 | $3
and immediate	andi $1, $2, 100	$1 = $2 & 100
or immediate	ori $1, $2, 100	$1 = $2 | 100
shift left logical	sll $1, $2, 10	$1 = $2 << 100
shift right logical	srl $1, $2, 10	$1 = $2 >> 100

Shift Operations

a. Shift amount : how many positions to shift

sll by i bits multiplies by 2^i

srl by i bits divides by 2^i

b. AND operations : useful to mask bits in a word.

select some bits, clears others to 0

consider: and $t0, $t1, $2

$t2	00	11	01	. . . .
$t1	11	11	00	. . . . .
$t0	00	11	00	. . . .

c. OR operations : useful to include bits in a word

set some bits to 1

or $t0, $t1, $t2

$t2	00	11	01
$t1	11	11	00
$t0	11	11	01

d. NOT operations : useful to invert bits

In MIPs , a NOR b = NOT (a OR b)

```
nor $t0, $t1, zero
```

$t1	0000	0000	0000	0000	0011	1100	0000	0000
$t0	1111	1111	1111	1111	1100	0011	1111	1111

* **Basic Blocks** : a sequence of ~~blocks~~ instructions w/ no embedded (except at end) branches, no target (except at beginning) branch

→ A compiler identifies basic blocks for optimization

→ An advanced processor can accelerate execution of basic blocks

* **Why is there no blt, bge etc?**

→ The hardware for <, ≥, ... is much slower than =, ≠.

→ Combining this with branch instructions involves more work per instruction, requiring a slower clock

→ beq and bne are the common case, which is a good design compromise.

* **Leaf Procedure** : a procedure that does not make any calls to other procedures (no nested calls)

* Data Transfer Operations

Instruction	Example	Meaning
load word	lw $1, 100($2)	$1 = memory [$2 + 100]
store word	sw $1, 100($2)	memory [$2 + 100] = $1
load upper imm	lui $1, 100	$ = 100 × 2^16

* Byte / Halfword Operations

for bitwise operations

lb rt, offset(rs)

lbu rt, offset(rs)

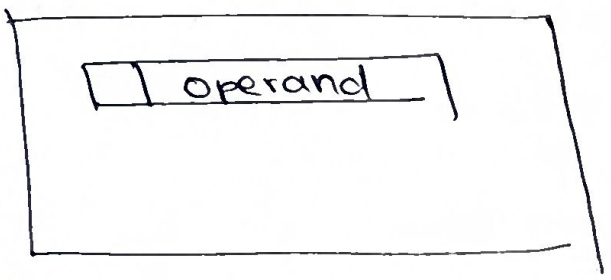
sb rt, offset(rs)

* Addressing mode Formats

① Immediate Addressing

→ operand is part of instruction

→ operand = address field.



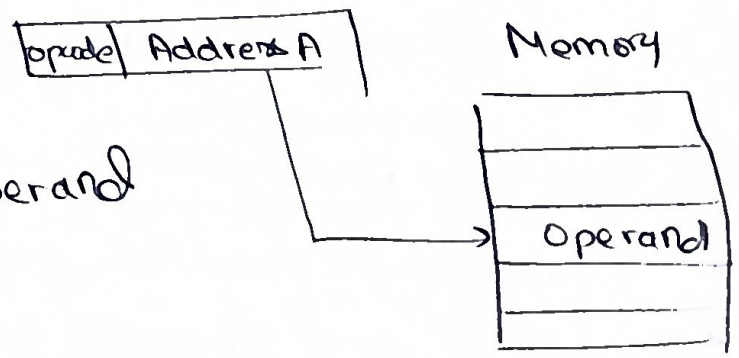
Pros & Cons : → no memory reference to fetch data

→ fast

→ limited range

② Direct Addressing

→ address field contains address of operand



for eg. ADD A

⇒ add contents of cell A to accumulator

⇒ look in memory at address A for operand

Pros & Cons → single memory reference to access data

→ no additional calculations to work out effective address

→ limited address space

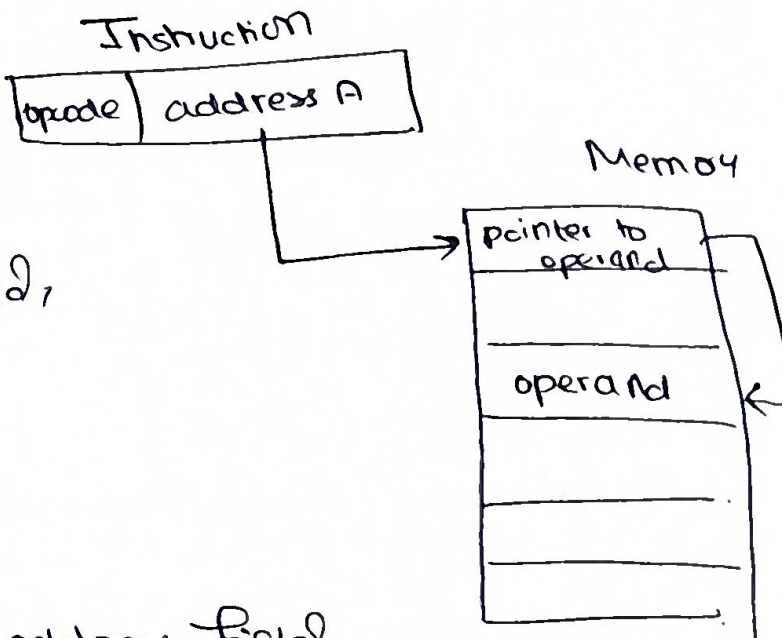
(3) Indirect addressing

→ memory cell pointed to by address field, which contains the address of the operand

→ effectively like using double pointers

eg. EA = [A]

look in A, find address (A) and look there for operand



Pros & Cons

→ large address space

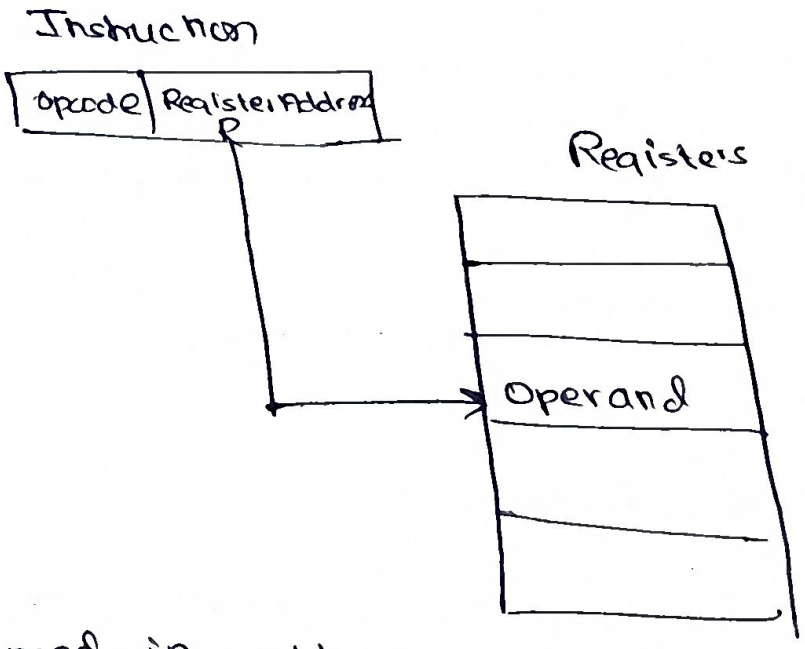
→ may be nested, multilevel, cascaded

→ multiple memory accesses to find operand, hence slower.

(4) Register Addressing

→ Operand is held in register named in address field.

EA = R



Pros and Cons

→ only a limited no. of registers

→ very small address field needed

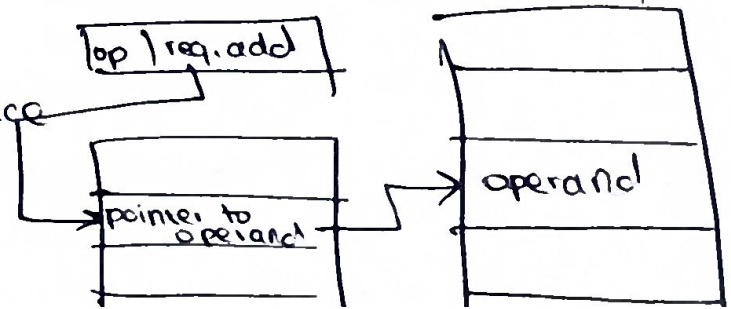
- shorter instructions
- faster instruction fetch

(5) Register Indirect Addressing

→ ~~Operand is held in register named in address field~~ → Operand is in memory cell pointed to by contents of register R.

→ EA = [R]

→ ~~Extended~~ Much larger address space



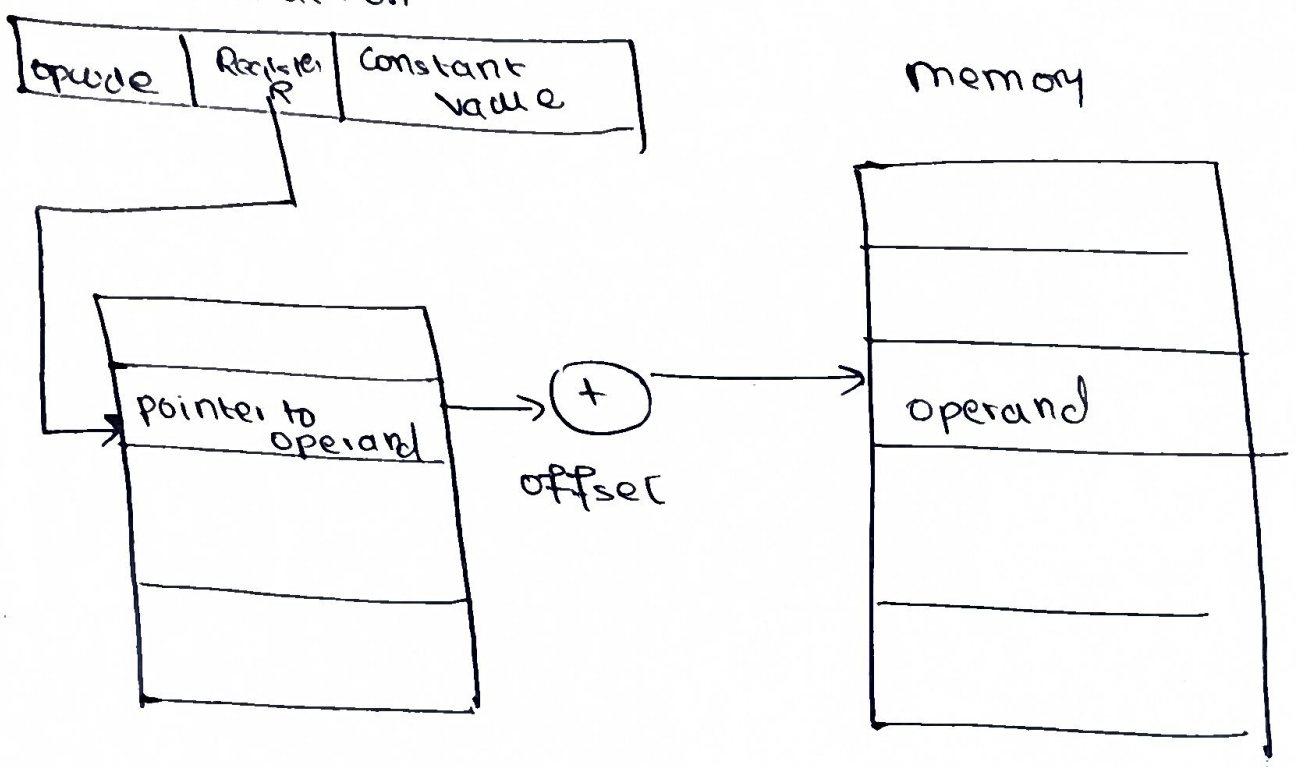
⑥ Indexed Addressing (Displacement Addressing)

→ EA = X + [R]

→ Address Field holds 2 values

X = constant value (offset)

R = register that holds address of memory



⑦ Relative Addressing

→ a version of displacement addressing

R = a program counter, PC

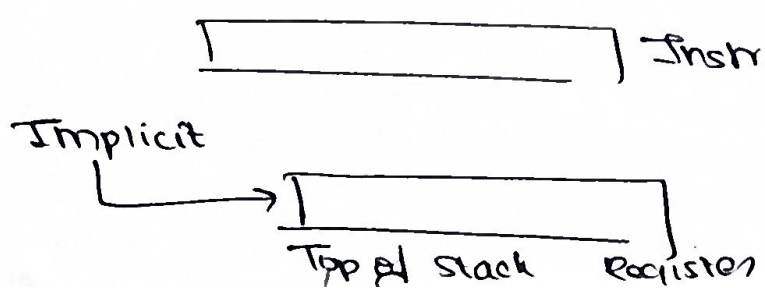
EA = X + (PC)

↳ i.e get operands X bytes away from current location pointed to by PC

⑧ Stack Addressing

→ operand is implicitly on top of stack

eg. ADD = pop top 2 items from stack and add and push



max no. of operations = 2^opcode

processor reg = $2^{\text{reg. field}}$

MBR = word length

MAR & PC = address length

Computer Organization and Architecture

Problems

Unit-1

(1) A computer uses a memory unit with 256K words of 32 bits each. A binary instruction code is stored in one word of memory. The instruction has 4 bits - an indirect bit, an operation code, and register code part to specify one of 64 registers, and an address part

(i) How many bits are there in the opcode, register code part & address part?

(ii) Draw the instruction word format and indicate the no. of bits in each part.

(iii) How many bits are there in the data and address inputs of the memory?

Solution

memory size = 256K = $2^8 \times 2^{10} = 2^{18}$

no. of registers = 64 = 2^6

word size = 32 bits

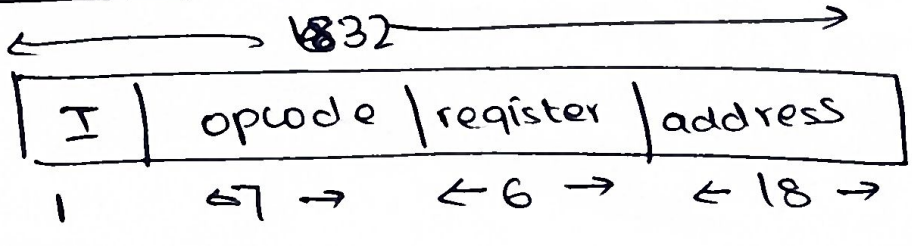
I → 1 bit

opcode → 7 bits

register → 6 bits

address → 18 bits

Instruction format



no. of data bits = word size = 32

no. of address bits = 18

(2) A memory unit with a capacity of 65,536 words of 25 bits each is used in a general purpose computer. The instruction code is divided into 4 parts: an indirect mode bit, opcode, 2 bits specifying a processor register & an address part.

(i) What is the max. no. of operations that can be in the computer if the instruction is stored in 1 memory word?

(ii) Draw the instruction word format, indicating the no. of bits and the function of each part.

(iii) How many processor registers are there in the computer & how many bits are there in each?

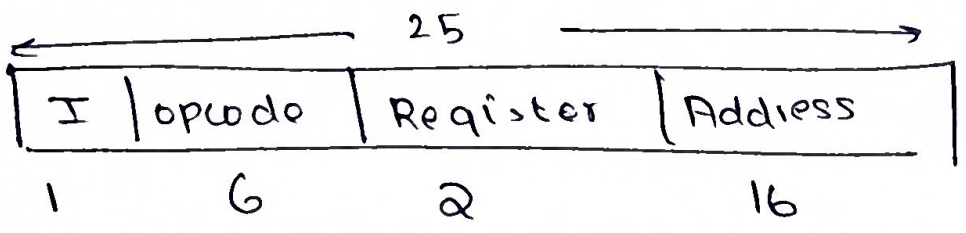
(iv) How many bits are there in MBR, MAR, PC ?

Solution: Bits to represent address = $65,536 = 2^{16}$

no. of used bits in a word = 25

bits to specify a processor register = 2

Instruction format



I = 1

opcode = 6

register = 2

address = 16

maximum no. of operations: $2^{(\text{no. of opcode bits})}$

$= 2^6 = 64$ operations

(iii) processor registers = 4

bits for each = 2 $\Rightarrow$ 00, 01, 10, 11

(iv) bits in MBR, MAR, PC

MBR → word length; MAR, PC → address length

MBR = 16

MAR = 16

PC = 16

(3) A 8 word instruction is stored in memory at an address specification by the symbol W. The address field of the instruction is stored at W+1, designated by Y. The operation used during the execution of an instruction is stored at an address symbolized by Z. An index register has the value X. State how Z is calculated for the following addressing types:

(a) direct $\Rightarrow$ Z = Y

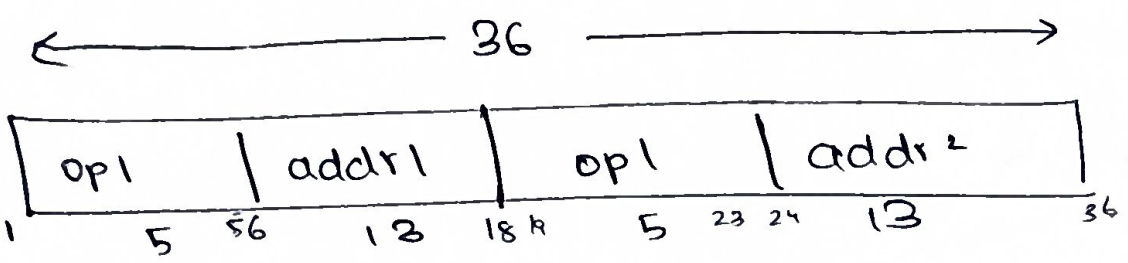
(b) indirect $\Rightarrow$ Z = M[Y]

(c) relative $\Rightarrow$ ~~Z = Y + X~~ (because index register) Z = PC + address = W + Y + 2

(d) indexed $\Rightarrow$ Z = Y + X

(4) A digital computer has a memory unit with a capacity of 8192 words, 36 bits per word. The instruct code format has 5 bits for the operation part & 13 bits for the address part. 2 instructions are placed in one memory word & a 36 bits IR. Formulate the instruction format & the fetch cycles

$8192 = 2^{13}$

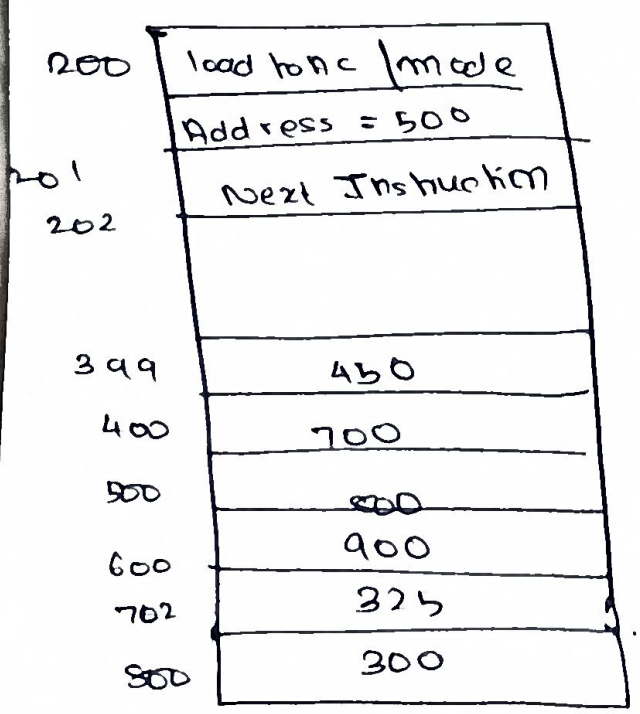


Fetch 1: MAR ← PC

Fetch 2 : MDR ← M , PC ++

Fetch 3 : IR ← MDR

(5) For the given memory structure, fill value for the diff. addressing types



$R_1 = 400$

$X_R = 100$

Addressing mode	Effective addr.	AC
(1) Direct addressing	500	800
(2) Immediate operand	201	500
(3) Indirect	800	300
(4) Relative PC + addr.	500 + 202 = 702	325
(5) Indexed XR + addr.	500 + 100 = 600	900
(6) Register direct		400
(7) Register indirect	400	700
(8) Auto increment (same as reg. indirect)	400	700
(9) Auto decrement	399	450

(6) Define Amdahl's law

$$\text{Speedup}_{\text{overall}} = \frac{\text{Old Exec Time}}{\text{New Exec Time}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enh}}\right) + \frac{\text{Fraction}_{\text{enh}}}{\text{speedup enh}}}$$

(7) Consider a hypothetical processor with an instruction of type lw R_1, 20(R_2), which during execution reads a 32 bit word from the memory and stores it in a 32-bit register R_1. The effective address of the memory location is obtained by the addn. of a constant to the register R_2. What is the addressing mode implemented?

Solution Base Indexed Addressing - implemented for arrays

(8) A 32-bit hex value B0A29C47 is stored at addr. location 1000. What the value of the byte in address 1002 in big endian & little endian format ?

big endian = 9C

little endian = A2

(9) A computer has a processor with a 24 bit address bus, 32 bit data bus and 8 control lines. Find the maximum amount of memory that the computer can address.

Size of address bus = 24 bits

Total size = 2^{24}

$= 2^4 \times 2^{20}$

$= 16MB$

(10) Write MIPs assembly for the following high-level statement

```
while (save[i] == k)
    i += 1
```

i in $s3

k in $s5

base address of save in $s6

Solution :

```
Loop: sll $t1, $s3, 2
      add $t1, $t1, $s6
      lw  $t0  0($t1)
      bne $t0, $s5, exit
```

4 bits for a word

sll = * 2

i=i*4

find addr of save[i], add w/base

load value

↑ do opp action

```
        addi   $s3   $s3 , 1
        j Loop
Exit :
```

(11) Consider a graphics card which is executing a program that consists of 50% of its total execution time is spent in doing floating point operations, and 20% of its total execution time is spent in floating point square root operations

Option 1: improve the FPSQR by a factor of 10

Option 2: improve FP by a factor 1.6

Which is better?

Answer

Option 1

$$\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{enh}) + \frac{\text{Fraction}_{enh}}{\text{speedup}_{enh}}}$$

$$= \frac{1}{(1 - 0.2) + \frac{0.2}{10}} = \frac{1}{0.82} = 1.22$$

Option 2

$$\text{Speedup} = \frac{1}{(1 - 0.5) + \frac{0.5}{1.6}} = \frac{1}{0.8123} = 1.23$$

Option 2 is better

(12) Compare computers R_1 and R_2, where R_1 has FP ops, while R_2 does not. Both computers have a clock frequency of 400MHz. The mixture of instructions is as follows:

command	share	R_1	CPI i / R_2
add	16%	6	20
mul	10%	8	32
div	8%	10	66
non-FP inst	66%	3	3

Calculate CPU and CPI for both computers

R_1

$$CPI = (0.16 \times 6) + (0.1 \times 8) + (0.08 \times 10) + (066 * 3)$$

$$= 4.54$$

R_2

$$CPI = (0.16 \times 20) + (01 * 32) + (0.08 \times 66) + (0.66 \times 3)$$

$$= 13.66$$

CPU Time

$$T = I \times CPI \times C$$

for R_1 $$T = 12000 \times 4.54 \times \frac{1}{400 \times 106} = 136.2 \times 10^{-6} \text{ s}$$

for R_2 $$T = 12000 \times 13.66 \times \frac{1}{400 \times 10^6} = 410 \times 10^{-6} \text{ s}$$

(13) Write MIPS assembly code for the following:

if $i < j$

then goto label

$i \rightarrow \$s1$

$j \rightarrow \$s2$

Solution

```
slt   $t0   $s2   $s1
beq   $t0   $zero   label
```

} remember to do the opp

(14) A computer has 456 inst. What should be the minimum width of the opcode field

$2^9 = 512 > 456$

$\Rightarrow$ min opcode size $= 9$

(15) If computer A has a clock rate of 200MHz & executes 30,000,000,000 instructions program in 2 minutes & computer B has a clock rate of 300MHz executes the same program in 3 minutes. Find the CPIs of A & B. Assume that they have the same ISA. Which computer is faster ?

$T = I \times CPI \times C$

$I = \frac{T}{CPI \times C}$

$T = I \times CPI \times C$

A

$$CPI = \frac{T}{I \times C} = \frac{2 \times 60}{3 \times 10^{10} \times \frac{1}{200 \times 10^6}} = \frac{2 \times 60 \times 2 \times 10^8}{3 \times 10^{10}} = 0.8$$

B

$$= \frac{3 \times 60}{3 \times 10^{10} \times \frac{1}{300 \times 10^6}} = \frac{3 \times 60 \times 300 \times 10^6}{3 \times 10^{10}} = 1.8$$

(11) $\frac{\text{Execution Time B}}{\text{Execution Time A}} = \frac{3}{2} = 1.5$

A is 1.5 times faster than B.

(16) An instruction is stored at location 300, with its address field at location 301. The address field has value 400. The content of 400 is 500. The processor register R_1 has the no. 200.

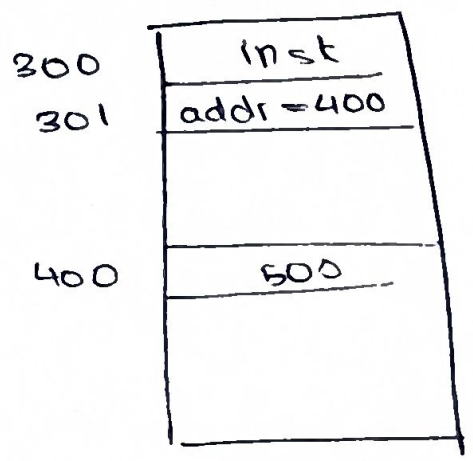
Evaluate the EA for the foll. addressing modes

(a) direct → 400

(b) indirect → 500

(c) relative → PC + addr = 302 + 400
= 702

(d) register indirect → 200



(e) indexed w/ R_1 as indexed register.
base addr + R
= 400 + 200 = 600

(17) Amdahl's law Questions.

$$\frac{1}{(1 - f_E) + \frac{f_e}{f_I}}$$

f_e → fraction enhanced

f_I = factor of improvement

Let a program have 40% of its code enhanced to run 2.3 times
What is the overall speedup?

$$S = \frac{1}{1 - f_E + \frac{f_E}{f_I}} = \frac{1}{(1 - 0.4) + \frac{0.4}{2.3}} = \frac{1}{0.6 + \frac{0.4}{2.3}} = 1.095$$