Page No.
Date 23 07

# Computer Organization

- **CPI Example :-**

Computer A : Cycle Time = 250ps , CPI = 2.0
Computer B : Cycle Time = 500ps , CPI = 1.2
Same ISA.
Which is faster & how many much?

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$
$$= I \times 2.0$$

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$
$$= I \times 1.2 \times 500\,ps = I \times 600\,ps$$

$$\frac{\text{CPU Time}_A}{\text{CPU Time}_B} = \frac{I \times 600\,ps}{I \times 500\,ps} = 1.2$$

- Clock Cycle = $\sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$
  [ If different instruction classes take different numbers of cycles ]

- Weighted Average CPI.

$$\text{CPI} = \frac{\text{Clock Cycle}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left( \text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{relative frequency}} \right)$$

- Alternative compiled code sequence using instructions in classes A, B, C.

| Class | A | B | C |
|---|---|---|---|
| CPI for class | 1 | 2 | 3 |
| IC in seq$^n$ 1 | 2 | 1 | 2 |
| IC in seq$^n$ 2 | 4 | 1 | 1 |

Sequence 1 : IC = 5
- clock cycle
= 2×1 + 1×2 + 2×3
= 10
- Avg. CPI = 10/5 = 2.0

Seqⁿ 2 : IC = 6
- clock cycle
= 4×1 + 1×2 + 1×3
= 9
- Avg. CPI = 9/6 = 1.5

Problem 1: A 400-MHz processor was used to execute a benchmark program with the following instruction mix & clock cycle count. Determin CPI, MIPS rate and execution time for program.

→

| Instruction type | Instⁿ count | Instⁿ clock cycle count |
|---|---|---|
| Integer arithmetic | 450000 | 1 |
| Data transfer | 320000 | 2 |
| Floating point | 150000 | 2 |
| Control transfer | 80000 | 2 |

$$= \frac{450000 \times 1 + 320000 \times 2 + 150000 \times 2 + 80000 \times 2}{10000000}$$

$$= \frac{450000 + 640000 + 300000 + 160000}{10000000}$$

$$= \frac{1550000}{10000000}$$

- Avg CPI = 1.55

- $PT = N \times CPI \times T$ / $TC \times CPI \times T$

$$= \frac{N \times CPI}{f}$$

$$= \frac{10000000 \times 1.55}{400 \times 10^6} = 3.87$$

- MIPS rate = $\frac{400 \times 10^6}{1.55 \times 10^6} = \frac{400}{1.55} = 258$ MIPS.

eg.2

| Instructⁿ Type | CPI | Instructⁿ mix |
|---|---|---|
| Arithmetic & logic | 1 | 60% |
| Load / store with cache hit | 2 | 18% |
| Branch | 4 | 12% |
| Memory reference with cache miss | 8 | 10% |

a) Calculate average CPI
b) Calculate the corresponding MIPS

→ a) $= \frac{1 \times 0.6 + 2 \times 0.18 + 4 \times 0.12 + 8 \times 0.1}{100}$

$= 2.24$

b) MIPS rate $= \frac{400 \times 10^6}{2.24 \times 10^6} = \frac{400}{2.24} = 178.57$ MIPS

$\frac{400 \times 10^6}{1.55 \times 10^6}$

eg.3 The execution times of four programs on three computers are given below:

| | Execution Time (in seconds) | | |
|---|---|---|---|
| Program | Computer A | Computer B | Computer C |
| prog 1 | 1 | 10 | 20 |
| prog 2 | 1000 | 100 | 20 |
| prog 3 | 500 | 1000 | 50 |
| prog 4 | 100 | 800 | 100 |

→

- MIPS Rate = $\frac{Ic}{T \times 10^6} = \frac{f}{CPI \times 10^6} = \frac{f \times Ic}{C \times 10^6}$

- Amdahl's law

Execution Time new = Execution time old $\times \left[ (1 - \text{fraction}_{\text{enhanced}}) + \frac{\text{fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right]$

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{fraction}_{\text{enhanced}}) + \frac{\text{fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$\boxed{So = \frac{1}{\left((1-f) + \frac{f}{s}\right)}} \quad \cdots \text{short form}$$

- Amdhal's law

Speed up = $\frac{\text{Performance for entire task using the enhancement when poss}}{\text{Performance for entire task using}}$

Speed up = $\frac{}{\text{Execution time for entire task using the enhancment when poss}}$

$f < 1$

$s > 1$

1. The fraction of the computational time in the original computer that can be converted to take advantage of the enhancement.

2. The improvement gained by the enhanced execution mode, that is how much faster the task would run if enhanced mode were for entire program.

Q1) Fraction enhanced = 0.4, speedup enhanced = 10,

speedup overall = $\frac{1}{\frac{0.6 + 0.4}{10}} = \frac{1}{0.64} \approx 1.56$

2) $S = 1.6 \quad f = 0.5$

$\frac{1}{\frac{(1-0.5) + 0.5}{1.6}} = \frac{1}{0.5 + 0.3125} = \frac{1}{0.8125} = 1.23$

$S = 10 \quad f = 0.2$

$\frac{1}{\frac{(1-0.2) + 0.2}{10}} = \frac{1}{0.8 + 0.02} = \frac{1}{0.82} = 1.22$

3) Frequency of FP operation = 25%
Average CPI of FP operation = 4.0
Average CPI of other instruction = 1.33
Frequency of FPSQR = 2%
CPI of FPSQR = 20.

ET = N × CPI × T

Average CPI of all FP operations is 2.5

Decrease CPI of FPSQR to 2

Solve By processor performance equation

$= \frac{N \times CPI}{f}$

CPI with new FPSAR = CPI original − 2%(CPI old FPSAR − CPI of new FPSAR only)

$= 2.0 - 2\% \times (20 - 2) = 1.64$

CPI new FP = (75% remaining × 1.33) + (25% × 2.5) = 1.625

Speedup new FP = $\frac{\text{CPU time original}}{\text{CPU Time new FP}}$

$= \frac{IC \times \text{clock cycle} \times CPI \text{ original}}{IC \times \text{clock cycle} \times CPI \text{ new FP}}$

$= \frac{CPI \text{ original}}{CPI \text{ new FP}}$

$= \frac{20}{1.625} = \frac{1}{1.23} = 0.813$

4) Speedup portion is 2 ; $f = 40\% \approx f = 0.4$ ; $f = 99\% \approx f = 0.99$

a) Speedup $= \frac{1}{0.6 + \frac{0.4}{2}} = \frac{1}{0.6 + 0.2} = \frac{1}{0.8} = 1.25$

b) Speedup $= \frac{1}{1 - 0.99 + \frac{0.99}{2}} = \frac{1}{0.01 + 0.495} = \frac{1}{0.505} = 1.98$

c) 40% of first application

$\frac{1}{0.2 + 0.8 \times 0.6 + 0.8 \times \frac{0.4}{2}} = \frac{1}{(0.2 + 0.8 \times 0.2 + 0.6 \times 0.4)} = \frac{1}{0.2 + 0.48 + 0.16} = \frac{1}{0.84} = 1.19$

$= \frac{1 \times 0.8 \times 0.2}{2} = \frac{1}{0.16} = 0.25$

d) 90% second application

$\frac{1}{0.8 + 0.2 \times 0.1 + 0.2 \times \frac{0.99}{2}} = \frac{1}{(0.8 + 0.002 + 0.099)}$

$= \frac{1}{0.901}$

$= 1.1098$

1. a) What is the speedup with N processor if 80% of the application is parallelizable, ignoring the cost of communication?

F = 0.8 S = N

$\frac{1}{1 - 0.8} = \frac{1}{0.2 + \frac{0.8}{N}}$ $S\infty = \frac{1}{(1-f) + \frac{f}{5}}$

b) $\frac{1}{(1 - 0.8) + \frac{0.8}{N}} = \frac{1}{0.2 + \frac{0.8}{N}}$

b) $\frac{1}{0.2 + 8 \times 0.005 + \frac{0.8}{8}} = \frac{1}{0.2 + 0.04 + \frac{0.8}{8}} = \frac{1}{0.2 + 0.04 + 0.1} = \frac{1}{0.34}$

$= 2.941$

c) $\frac{1}{0.2 + 3 \times 0.005 + \frac{0.8}{8}} = 3.17$

d) $\frac{1}{0.2 + \log_2 N \times 0.005 + \frac{0.8}{N}}$

e) $\frac{d}{dN}\left(\frac{1}{\left(1 - \frac{p}{100}\right) + \log N \times 0.005 + \frac{p}{N}}\right) = 0$

$$S = \frac{ET_{up}}{ET_p}$$

7) Average instruction execution time = clock cycle x Average CPI
= 1ns x [(40% + 20%) x 4 + 40% x 5]
= 1ns x 4.4 = 4.4ns

Speedup from pipelining = Average instruction execution time of up / pipeline
= 4.4ns / 1.2ns = 3.7 times

10) a) What is the clock cycle time of the 5-stage pipelined machine?
→ 2 ns + 0.1 = 2.1 ns

b) Stall every 4 instruction, what is the CPI of the new machine?
→ 5 cycles / stages / 4 instruction = 1.25

c) Speedup of pipelined machine over single cycle machine?
→ I x CPI x time original
I x CPI x new

ET = I x CPI x Cycle time
Speedup = (I x 1 x 7) / (I x 1.25 x 2.1) = 2.67

d) I x 1 x 7/5 x 1 x 0.1 = 7.0 Ignoring extra stall cycles it would be

Kai Hawang Pdf. pg 135

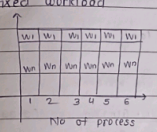
is based on a fixed workload or a fixed problem size:

Amdahl's law :- Eq 0.13

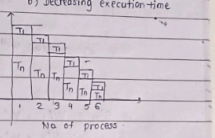
$$S_n = \frac{n}{1 + (n-1)\alpha}$$

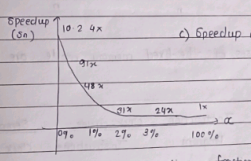
$W_i = \alpha$ $W_n = 1 - \alpha$ if
$W_n = W_i = 0$ for $i \neq 1$ & $i \neq n$

a) Fixed workload




b) Decreasing execution time



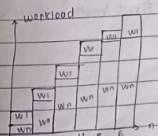




c) Speedup with fixed load

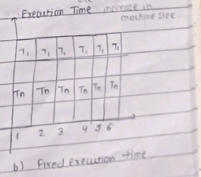
sequential diagram fraction of program

Kai Hawgan pg 137

Fixed-time second model and Gustafson's law: * is applied to scaled problems where problem size increases with machine size




a) Scaled workload




b) Fixed execution time

Page No.
Date

speedup
Sn
1024x
1014x 1004x 993x 983x
$S_{1024} = 1024 - 1023x$
0% 1% 2% 3% 4% ∞

c) Speedup with fixed load

pg. 140 kaibook

$$S^*_n = \frac{W^*_1 + W^*_n}{W^*_1 + \frac{W^*_n}{n}} = \frac{W_1 + G(n)W_n}{W_1 + \frac{G(n)W_n}{n}} \quad (9.33)$$

Special cases of the fixed memory model are Amdal's & Gustafson's law.

CPU Time = $\frac{\text{Instruction}}{\text{Program}} \times \frac{\text{clock cycles}}{\text{Instructions}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$

Read : 1st Chapter of William 9th / 10th / 11th

Q Intel is variable length or fixed length?

Page No.
Date 05 08 24

1st address

3) op
S↑
op → operation Eg Z = K + B * C
100 10 5

1) op d , 2nd address → op Z ,
d
dest$^n$ operand
MPY R1,C
ADD R2,B
MPY R1,R2,R3 3 add
ADD Z,R1,K

2) op d , 2nd add a) zero
↑ S↑ op↑
MOV R1,C
MUL R1,B 2 address
ADD Z,R1K
MOV Z,R1

a) One address instructions

| Instn | Comment : ← accumulator |
|---|---|
| LOAD D | AC ← D |
| MPY E | AC ← AC x E |
| ADD C | AC ← AC + C |
| STOR Y | Y ← AC |
| LOAD A | AC ← A |
| SUB B | AC ← AC − B |
| DIV Y | AC ← AC ÷ Y |
| STOR Y | Y ← AC |

LOAD C
MUL B
ADD E
STORE Z
1 address

K,B,C 24 bit
push K
push B
push C
MUL
ADD
POP Z
0 address

b) Two address instruction

| | |
|---|---|
| MOV Y, A | Y ← A |
| SUB Y, B | Y ← Y − B |
| MOVE T, D | T ← D |
| MPY T, E | T ← T x E |
| ADD T, C | T ← T + C |
| DIV Y, T | Y ← Y ÷ T |

Three address

| | | Comment |
|---|---|---|
| SUB | Y, A, B | Y ← A - B |
| MPY | T, D, E | T ← D x E |
| ADD | T, T, C | T ← T + C |
| DIV | Y, Y, T | Y ← Y ÷ T |

program to Execute Y = (A - B) / (C + (D x E))

AB - CDE ÷ + *

Zero address

Push A
Push B
SUB
DIV
Push C
ADD
Push D
Push E
MUL
Pop Y

eg Z = (((P - (Q + S)) * T) / (U / (V - W)))

| Three address | | | Two address | |
|---|---|---|---|---|
| ADD | Z, Q, S | Z = Q + S | MOV | Z, Q | Z ← P |
| MUL | Z, Z, T | Z = Z x T | SUB | Z, S | Z = Z - S |
| SUB | Z, P, Z | Z = P - Z | MOV | R, P | |
| SUB | T, V, W | T = V - W | SUB | R, Q | |
| DIV | T, U, T | T = U ÷ T | MUL | R, T | |
| DIV | Z, Z, T | Z = Z/T | MOV | R, V | |
| | | | SUB | R, W | |
| | | | MOV | R2, U | |
| | | | DIV | R2, R0 | |
| | | | DIV | R1, R2 | |
| | | | MOV | Z, R1 | |

One address

| | |
|---|---|
| LOAD Q | STORE Temp1 |
| ADD S | LOAD U |
| MUL T | DIV Temp1 |
| ST Temp | |
| MUL T | |
| ST Temp | |
| LOAD P | STORE Temp1 |
| SUB Temp | LOAD Temp |
| STORE Temp | DIV Temp1 |
| LOAD V | STORE Z |
| SUB W | |

Op | Operands

1 | 2 | 3

16 bit R & 16 bits

14 of type 3 add inst ✓ — 14 x 3 12 bit for operands, 4 bits for opcode
30 of type 2 —1—
28 of type 1 —1—
64 of type 0 —1—

escape code 1111111111 0000 to 63 = 64

0000 _01_ _02_ _03_

1101 _____ _____ _____

1110 0000 _____ _____ _____ operand 8 bits opcode 4 bits opcode

escape code

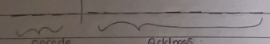
1111 11101

4 bits operand 12 bits opcode
000 to 27 12 out 5 gone
left hand 7
remaining 5 rem's

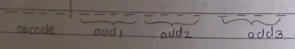
By solving this all we can add to get total number of address type 3 - $14 \times 2^{12} + 30 \times 2^{8} + 28 \times 2^{4} + 64 = 65536 = 2^{16}$

$2^{0} 2^{4} 2^{8} 2^{12}$

Ex:) Consider a machine with 16 bit instructions and 16 register.
- The instruction format can have several structures:
  - Opcode + Memory address (such as MARIE):
→ If we have 4KB byte addressable memory we need 12 bits to specify an address location.
→ The remaining 4 bits are used for the opcode: 16 instructions are hence available.




- Opcode + Register Addresses:
→ we need 4 bits to select one of the 16 bit available register.
→ Suppose we have 4 bits




eg:2) Consider a machine with 16 bits instructions 16 register. Find out such to encode the following instruction:
- 15 instr with 3 address
- 14 2 add
- 31 1 add
- 16 0 add

Can we encode this instruction set in 16 bits?
→ Yes, if we use expanding opcodes.

0000 R1 R2 R3 } 15 - 3 $15 \times 2^{12} + 14 \times 2^{8} + 15 \times 2^{12} = 61490$ bit
1110 R1 R2 R3

1111 0000 R1 R2 } 14 2 $14 \times 2^{8} = 3584$ bit
1111 1101 R1 R2

1111 1110 0000 R1 } 31 1 $31 \times 2^{4} = 496$ bit
1111 1111 1110 R1

1111 1111 1111 0000 } 16 0 $61440 + 3584 + 496 + 16 = 16 \times 2^{16} = 2^{16} = 65536$
1111 1111 1111 1111

eg:3) Is it possible to design an expanding opcode to allow the following to encode with a 12 bit instruction? Assume a register operand requires 3 bits.

4 inst 3 regis $4 \times 2^{9} + 255 \times 2^{3} + 16 = 8192$
255 inst 1 reg 4104
16 inst 0 reg

Q.4) Assume the CPU having 16 registers and inst$^{n}$ length is 16 bits it should be having 14 of type - 3

31 2
12 1
Calculate at the most how many number of type 0 instruct it should have. Show appropriate encoding → 64

→ 0000 R1 R2 R3 } 14 $14 \times 2^{12} + 31 \times 2^{8} + 12 \times 2^{4} + 64$
1101 R1 R2 R3 65536

1110 0000 R1 R2 } 31
1111 1110 R1 R2

1111 1111 0000 R1 } 12
1111 1111 1011 R1

1111 1111 1111 0000 } 64
1111 1111 1111 1111

① 14 instruction 3 register type 3

0-13 0000 R1 R2 R3 $2^4 \times 2^4 \times 2^4 = 2^{12} \times 14$
1101 R1 R2 R3

0-30 31 instruction 2 register 2 type

1110 0000 R1 R2 $2^4 \times 2^4 = 2^8 \times 31$
1111 1111 R1 R2

0-11 12 instruction 1 register type 1
1111 1111 0000 R1 $2^4 \times 12$
1111 1111 1100 R1

64 instruction 0 register type 0
1111 1111 1111 0000 $64 \times 2^0$
1111 1111 1111 1111

① 14 inst$^r$ typ 3 16 bits 4 bytes
30 inst$^r$ type 2
28 inst$^r$ type 1
64 inst$^r$ type 0

0-13 0000 R1 R2 R3 } 14 3 $14 \times 2^{12} + 30 \times 2^8 + 28 \times 2^4 + 64$
1101 R1 R2 R3 $= 65536$

0-29 1111 0000 R1 R2 } 30 2
1111 1101 R1 R2

0-27 1111 1110 0000 R1 } 28 1
1111 1111 1011 R1

1111 1111 1111 0000 } 64 0
1111 1111 1111 1111

12 bit 3 bytes
4 3
255 1
16 0

0000 R1 R2 R3 } 4 3
0000 R1 R2 R3

100 000 000 R1 } 255 1
1111 1111 1111 R1

1111 1110 000 } 16 - 0
111 111 111

Q 4] Is it

7 inst$^{n}$ = 2 15 bit addr & 1 3 bit reg

Soo 1 15 bit addr & 1 3 bit reg

So inst$^{n}$ with no address or reg

→ $7\times2^{15}\times2^{15}\times2^{3} = 7\times2^{33} = 60,129,542,144$ bit pattern.

· next soo inst$^{n}$

So bit pattern

· total we need $6.026\times10^{10}$ bit patterns.

· with 36 bit inst$^{n}$ we can have $2^{36} = 6.87\times10^{10}$ bit pattern.

· We have enough bit patterns so it can create opcodes

Q A processor has 16 bit inst$^{n}$ length, 8 reg & memory address of 8 bit. Is it possible to have following types of instructions:

→ = inst$^{n}$ of type that have 2 reg fields & 1 mome addr

→ It is possible

| | |
|---|---|
| 00 | |
| 01 | |
| 10 | |

Q 6 number of inst$^{n}$ of type that have 1 reg field & 1 meme address

0000 11 | 101 |

Q 6 number of inst$^{n}$ of type that have all 3 reg field &

111 | 000 R1 R2 R3

111 | 101

It is possible to accomodate

Memory pattern - 5th edition

CO

Q. 3 inst$^{n}$ of type that have only single memory address as it fields

111111 0000 R1 R2

111111 0010 R1 R2

Q. 7 no of inst$^{n}$ of type that have two register fields

1111 | 0000 R1 R2

1111 | 0110 R1 R2

Q 7 no of inst$^{n}$ of type that have only one register field

0000 R1

0110

111111111 | 000

111111111 | 110

Q 8 no of instructions of type that have zero address.

111111111111 | 000

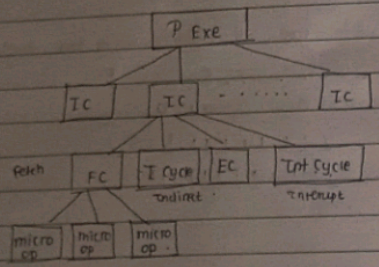
111111111111 | 111

Date 12 08 24

Control Unit: William Stalling chapter 20

- There are 4 particular sub-cycles.
  1) Instruction cycle
  2) Indirect cycle
  3) Execution cycle
  4) Interrupt cycle

Microoperation - that are functional or atomic operation.
micro refers to simple and complete small or little work of task.



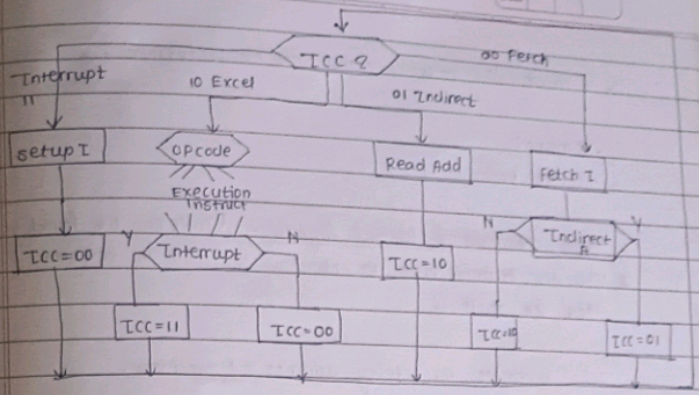

Fetch cycle - It occurs at beginning of each Inst^n cycle. and cause an intst^n to be fetch from memory.

1) MAR - Memory Address Register.
2) MBR - Memory Buffer Register.
3) Program counter - PC
4) IR - Instruction register

→ Register Involved in Fetch

MAR ∈ (PC) must percept. MBR ∈ Memory
MBR ∈ (memory and IR ∈ (MBR) not in same section

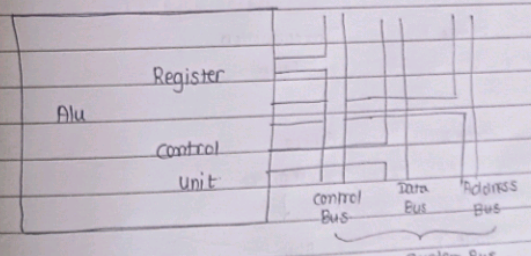
IR :- Once the instruct^n is fetched it fetched source operands.




Control Unit Functional Requirements. 13-08.

- The control unit performs two basic task:
  1) Sequencing micro operations.
  2) Execution.

figure - The CPU with System Bus.




Three steps to process to lead characteristic of control unit?
1) Define basic elements of the processor
2) Describe micro-operations processor performs.
3) Determine the function that are the control unit must perform

Hamacher Zvonko & Zaky

William stalling
control unit operation Ch 7
716 pg
Hoboken

to cause the micro-operations to be performed.

- Datapath
- Control path

- Micro-operations & Control Signals (Table can be found on pg no 719 in textbook)

@ Why the excution cycle is not there in table ?

Who generates the memory address → processor

- Instruction Execution chapter 4 - The processor - 3

1st and 2nd from William stawling 20

performance

17/03/24

- Bus Processing Unit

A five stages organization

Stage 1 Introduction fetch

Stage 2 Source register

↓

Stage 3 ALU

↓

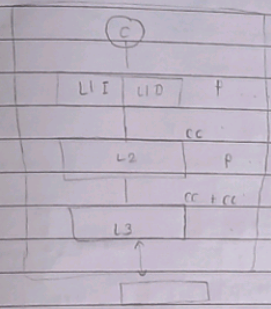
4 Memory access

↓

5 Destination register

Hennesy patterson
RISC-V-Edition 4.1-4.5

@ What is block size and who decide it?
@ What is latency of ram?

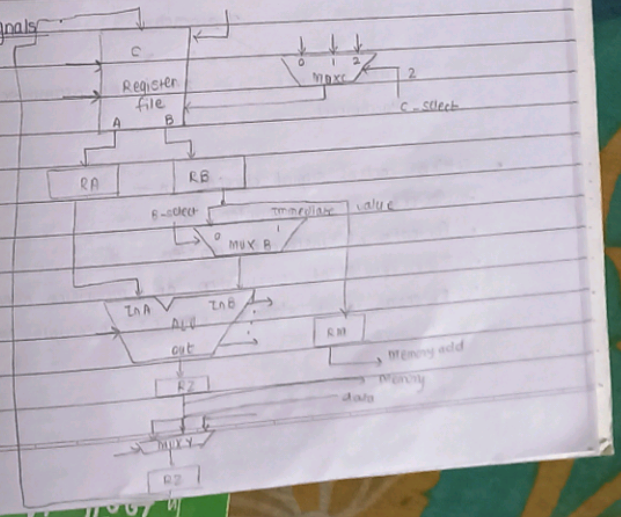



Branch_if_[R5] = [R6] LOOP Figure 5.16 Chapter 5 (zaky) no 19

Call_Register R9 5.17

- Waiting for Memory

5.5 Control Signals



Page No.
Date

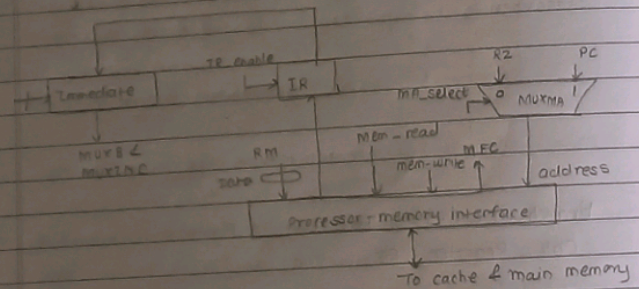
Q. From where return address are coming?

5.19 figure 5.20 figure

possibility: 1. PC = PC + 4

2. Branch target inst$^n$

3. Subroutine




- Hardwired Control

2 approach

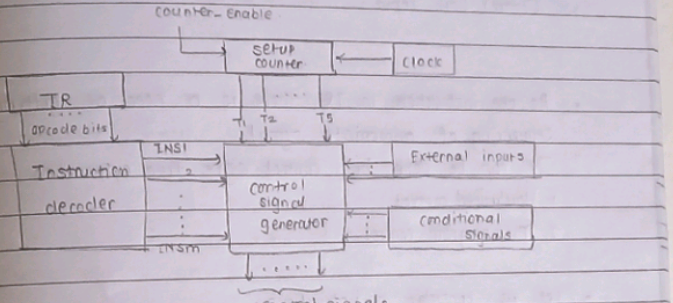
1. Hlw control (Hardwired)
2. Microprogrammed Control

- Setting control signal depends on —
  - Content of step counter
  - Contents of instruction register
  - The result of comparision operation
  - External input signals, such as interrupt requests.

Page No.
Date

Fig 5.21 $T_1 - T_5 \rightarrow$ IF, ID, IE, M, WB

control signal sequencing & — execution




5.6.1 • Datapath control Signals.

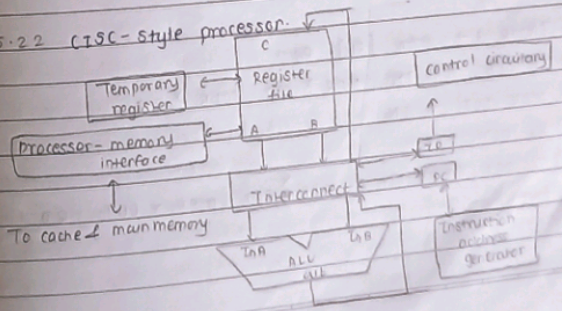
RF write = T5 · (ALU + Load + call) controlling datapath

Counter_enable = $\overline{WMFC}$ + MFC

↳ wait for a memory operation to be completed

PC_enable = $T_1 \cdot MFC + T_3 \cdot BR$

5.22 CISC-style processor.



Page No.
Date 26 08 24

Table 20.3 A Decoder with 4 Inputs and 16 Outputs.
(William Stalling)

Control Unit Input

- As per instruction in IR there is a need of getting sequencing of appropiate signal
  It can be done by ~~control signe~~ two approches
  1. Hardwained control
  2. Microprogram control

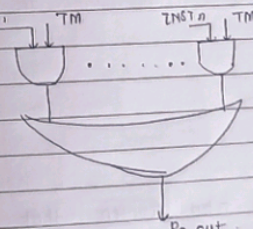
- Let us consider 3 instructions
  1. INSTR1 → add R1, R0
     Execute this : i) R1 OUT, x IN
     ii) R0 OUT, Add
     iii) Z OUT, R1 IN
     iv) END
  2. INSTR 2 → XOR R0, R5
     i) R0 OUT, x IN
     ii) R5 OUT, XOR
     iii) Z OUT, R0 IN
     iv) END
  3. INSTR 3 → ADD [R0], R6
     i) R0 OUT, MAR IN, MAR * OUT, MDR * IN
     ii) MDR OUT, x IN
     iii) R6 OUT, ADD
     iv) Z OUT, MDR IN, MDR * OUT, MAR x OUT, RD
     v) END

Page No.
Date

R0 out = ((INSTR AND T2) OR
( AND T1) OR
( AND T1) OR
.....).

Where INSTR1 and T2 means R0 out should be made 1 if there is instruction 1 and second clock cycle
ADD = INSTR1 and T2 and T3) OR .....
:
END.
INSTR1 and T4 or INSTR1 and T2

The and combination is for activation of particular signal in context of single instruction.
Here, for every instruction there is one and gate and giving to each and gate one timing marker and instruction marker
All AND gates are OR.




We required two types of signals for this purpose—
1) Time marker
2) Instruction marker

Chapter 20 william stawling

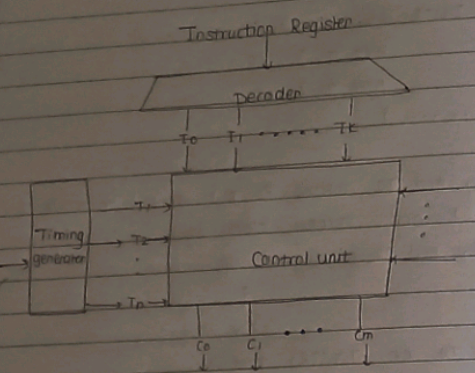



fig Contro Unit with Decoded Inputs.

1) Different instruction with different steps which means naturally diff no of clock cycles to execute.
At the most 8 bit required choose 3 bits.
If 8 steps 3 bit counter
If 16 steps 4 bit counter.

00 $T_0$
01 $T_1$
10 $T_2$
11 $T_3$

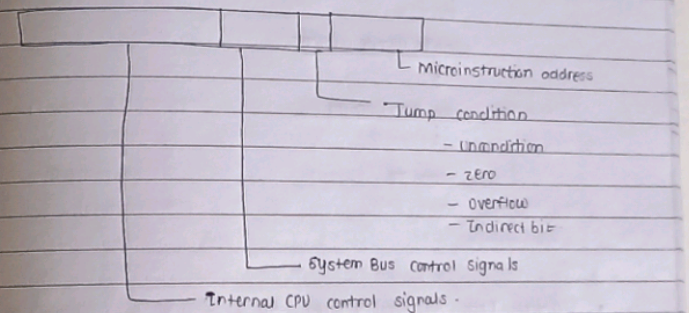
2) End signal connected to n bit counter that indicates one instruction is completed and set n bit counter to (quater) 0000
3) After the previous step now IR is loaded with new instruction and counter starts counting the clock.
4) The approch discussed in hardware based control unit design lacks till we throw the design or scrab the design.
No flexibility for change as it is permanetly implemented in hardware architecture.

27-08

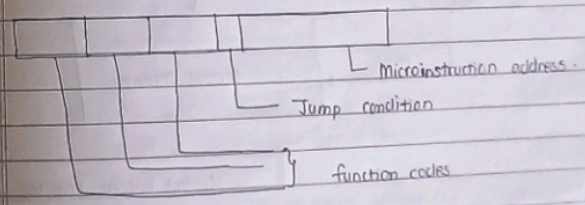
Chapter 21 William Stawling

- Machine Instruction set for wilkes example Table 21.1

a) Horizontal Microinstruction




b) Vertical Microinstruction.




Control Memory

Fig 21.2 Organization of Control Memory

:

jump to execute

:

jump to execute

:

jump to fetch

:

jump to opcode routine

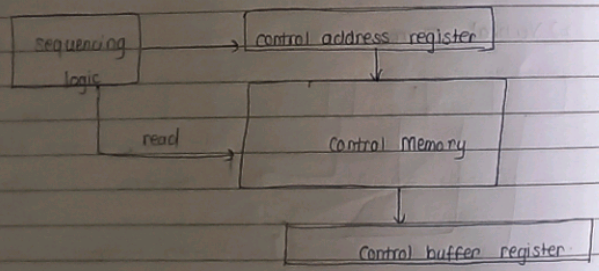
:

jump to interupt

:

jump to

- Control Unit Microarchitecture fig 21.3.




Control unit function are as follows:

1) To execute an instruction, the sequencing logic unit issues a READ command to the control memory.
2) The word whose address is specified in the control address register is read into the control buffer register.
3) The content of the control buffer register generates control signals and next-address information for the sequencing logic unit.
4)

- fig 21.4 Functioning of Microprogrammed Control unit.

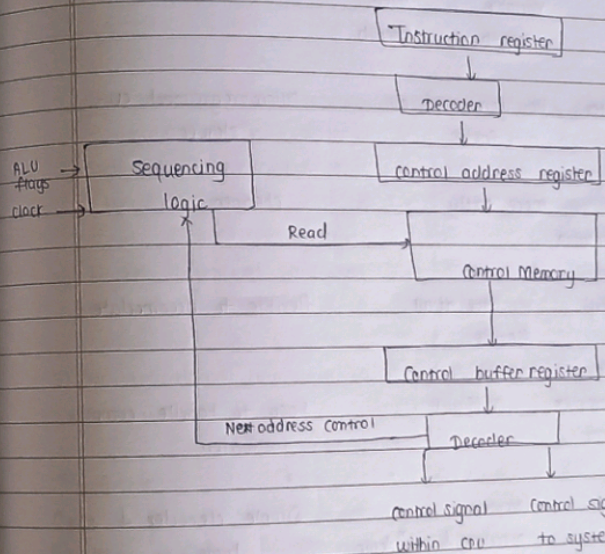



Table 21.2 (Read).

- Microinstruction sequencing.
  Design Consideration.

Fig. 21.6 Branch control Logic : Two Address fields
Fig 21.7 —1— : Single Address fields

Fig 21.8 ——1—— : Variable Format

Microprogram

Sequencing & CISC — Variable length

RISC — Fixed Length

| Attributes | Hardware based CU | Microprogramed CU |
|---|---|---|
| 1) speed | faster | slower. |
| 2) Cost of implementation | more costly | cheaper |
| 3) Flexibility | Not at all flexible | flexible to accomodate |
| 4) Ability to handle complex instructions | Difficult | Easy to handle complex instruction |
| 5) Decoding | Complex Decoding & seq$^n$ logic | Simple decoder & seq$^n$ logic. |
| 6) Applicability | Used by RISC kind of processor | Used by CISC kind of processor. |
| 7) Instn$^n$ set size | Smaller | Larger |
| 8) Control memory | absent | Present |
| 9) Cheap area requirement | less because control memory absent | more because control |
| 10) Occurance of error | more | less |

| Horizontal Microinstruction | Vertical Microinstruction |
|---|---|
| 1) Supports longer control word | Supports shorter control word. |
| 2) Higher degree of parallisum | Allows low degree of parallisum ie degree is either 0 or 1. |
| 3) No additional hardware is required | Additional hardware is required in the form of decoders |
| 4) faster | slower. |
| 5) less flexible | more flexible |
| 6) less use of ROM encoding as compared to vertical. | makes more use of ROM encoding to reduce length of control word. |

Page No.
Date 2 09 24

# Chapter 2

## Arithmetic for Computers

- Integer Addition

eg 7+6

| (0) | (0) | (1) | (1) | (0) | (carries) |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | |
| 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | (1) |

Overflow
- Adding +ve and -ve operands, no overflow
- Adding two +ve no. MSB is 1 is overflow
- Adding two -ve no. MSB is 0 is overflow

- Integer Subtraction

Add negation of second operand

eg 7-6 = 7+(-6)

| | | |
|---|---|---|
| +7 | 0000 0000 ... 0000 0111 | |
| -6 | 1111 1111 ... 1111 1010 | |
| +1 | 0000 0000 ... 0000 0001 | |

0 -ve
1 -ve

Overflow if result out of rang
- subtracting two +ve or two -ve operands, no overflow
- subtracting +ve from -ve operand overflow if result sing is 0
- subtracting -ve from +ve operand overflow result is 1

Page No.
Date

- Arithmetic & Logical Unit

- Integer Representation
  - binary number system
    - The digits zero and one
    - The minus sign (for negative)
    - The period or radix point

- Sign - Magnitude Representation

- Table 10.1
Characteristics of Twos complement Representation and Arithmetic

| | |
|---|---|
| Range | $-2^{n-1}$ through $2^{n-1}-1$ |
| Number of Representations of zero | one |
| Negation | Take the boolean complement of each bit of corresponding positive number, then add 1 to the resulting bit pattern viewed as an unsigned integer |
| Expansion of Bit Length | Add additional bit position to the left and fill in with the value of the original sign bit |
| Overflow Rule | If two no. with same sign are added then overflow occurs if and only if the result has the opposite sign |
| Subtraction Rule | To subtract B from A, take the twos complement of B and add it to A |

sign (-ve) → 1001 → -1

- Table 10.2
Alternative Representations for 4-Bit Integers

| | | 2's cm |
|---|---|---|
| +1 | 0001 | 0001 |
| -1 | 1001 | (0001+1) 1111 |

- Range Extension

- In sign-magnitude notation this is accomplished by moving the sign bit to the new leftmost position and fill in with zeros.

- The procedure

1) Rule is to move the sign bit to the new leftmost position and fill it with copies of the sign bit
2) For ~~position~~ positive no.'s fill with zeros
3) for -ve numbers, fill it with 1
This is called sign extention.

- Fixed Point Representation

Negation Special Case 1

0 = 00000000 (twos complement)

Bitwise Complement = 1111 1111

Addition + 1

Result 10000000

Case 2

-128 = 1000 0000 (twos comple

Bitwise complem 0 111 1111

Add 1 to LSB + 1

1000 0000

✓ 21.1 to 21.3 - william stawing

So. -(-128) = -128 ×

Monitor MSB (sign bit)

It should change during negation

07-09-17

- Nega figure 10.3 Addition of Numbers in twos Complement Representation.

a) 1001 = -7 + 0101 = 5 → 1110 = -2

b) 1100 = -4 + 0100 = 4 → 1000 = 0

c) 0011 = 3 + 0100 = 4 → 0111 = 7

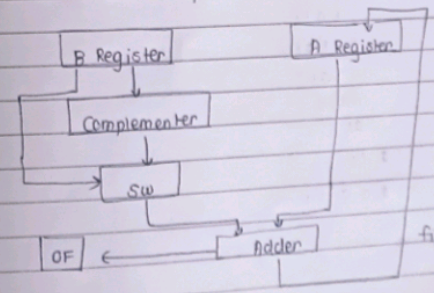
d) 1100 = -4 + 1111 = -1 → 1011 = -5

e) 0101 = 5 + 0100 = 4 → 1001 = overflow

f) 1001 = -7 + 1010 = -6 → 0011 = overflow (avoid)

Overflow Rules - two no. with same sign but result is of different / opposite sign.

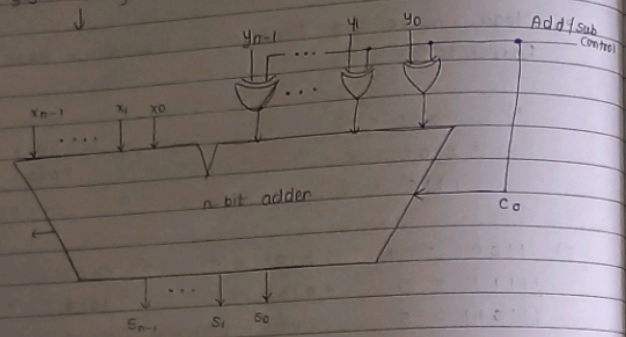
- fig 10.4 william
- fig 10.5 Gemotric Depiction of Twos Complement Integers. william
- fig 10.6



9.1.1 Addition / Subtraction Logic Unit

fig 9.2 Logic for addition of binary numbers.

fig 9.3 Binary addition / subtraction logic circuit.




9.2.1 Carry-lookahead addition

diag (a) - Bit ~~carry~~ Bit stage cells

(b) 4-Bit adder

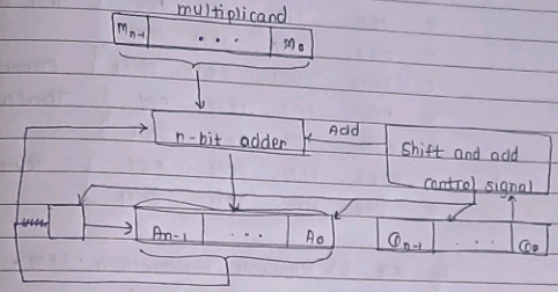
9.4 Ref fig 9.5

8-09-24

10.7 Multiplication of Unsigned Binary Integer.

```
  1011
 x1101
 ------
  1011
 0000
1011
1011
--------
10001111
```

10.8 Hardware Implementation of Unsigned Binary




(a) Block diagram

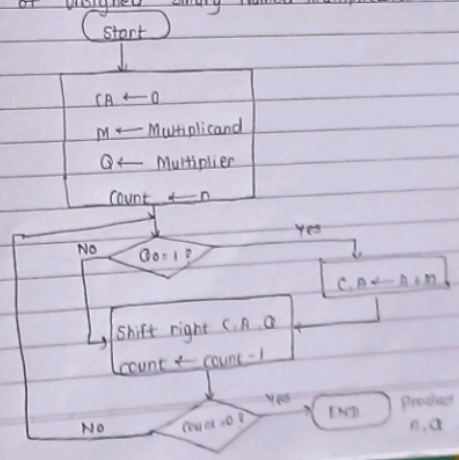
M - multiplicant register

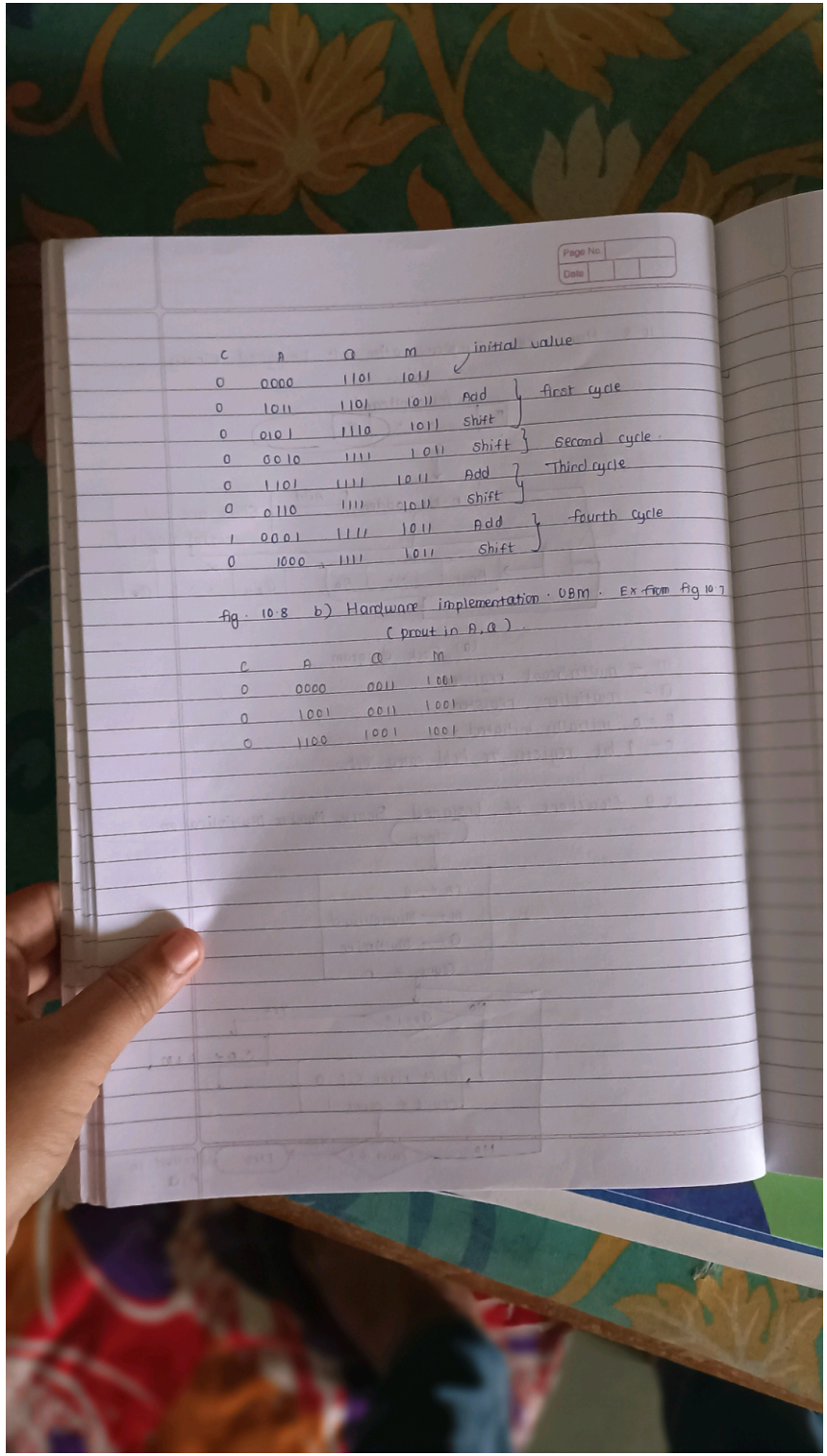
Q - multiplier register

A = 0 initially initiated

C - 1 bit register to hold carry out

10.9 flowchart of Unsigned Binary Number Multiplication





| C | A | Q | M | |
|---|---|---|---|---|
| 0 | 0000 | 1101 | 1011 | initial value |
| 0 | 1011 | 1101 | 1011 | Add } first cycle |
| 0 | 0101 | 1110 | 1011 | Shift } |
| 0 | 0010 | 1111 | 1011 | Shift } Second cycle |
| 0 | 1101 | 1111 | 1011 | Add } Third cycle |
| 0 | 0110 | 1111 | 1011 | Shift } |
| 1 | 0001 | 1111 | 1011 | Add } fourth cycle |
| 0 | 1000 | 1111 | 1011 | Shift } |

fig. 10.8 b) Hardware implementation · UBM · Ex from fig 10.7 (prout in A, Q).

| C | A | Q | M |
|---|---|---|---|
| 0 | 0000 | 0011 | 1001 |
| 0 | 1001 | 0011 | 1001 |
| 0 | 1100 | 1001 | 1001 |