Sumedh

- Topics

- *Basic processing unit*
- *ALU operations*
- *Instruction execution*
- *Branch instruction*
- *Multiple bus organization*
- *Hardwired control*
- *Generation of control signals,*
- *Micro-programmed control*
- *Pipelining: Basic concepts of pipelining*
- *Performance*
- *Hazards-Data, Instruction and Control*
- *Influence on instruction sets.*

- Instruction Execution

1. Instruction Set processor (ICP) : The part CPU that reads n executes instructions

2. CPU : The main part

3. prog execution : progs run as a sequence of instructions

4. Execution : CPU executes

- fundamental Concepts

1 Instruction fetch : The CPU fetches 1 instruc at a time
2. PC : prog. counter that holds the address of the next instruction
3. IR : holds instruc abt current execution

Steps to execute :-

① Retrieve info in PC & load to IR
② PC increments by 4 bytes
③ Perform the op
④ CPU performs it using data transfer, ALU & mmry access

- Single bus organisation

Parts

1. ALU
2. Regs - Store
3. digital circuits - gates
4. Internal path - to transfer data
5. Driver Circuits - transmit signal
6. Receiver Circuits - rec frm external circuits

Registers

- PC : Tracks add of next instrue
- MAR : Holds add of mmry loc
- MDR : Holds data being transferred to mmry
- Specific purpose regs : indexc / stack regs
- Temp regs : (Y, Z, temp)

Multiplexer : Select data to send to ALU, chooses blw a reg's output & a const increment

Data path : refers to ALU, regs & bus together. It is the path data travels thru

- Four Steps of Instruction Execution

1. Register Transfer

Signals : 1. Ri in - Control loading data in Ri frm bus
2. Ri out - " placing " in bus " Ri

Clock cycles : All processor data transfers happen within defined clck cycles

multiple clck sig also used

data transfer : eg move data R_1 to R_4

R_1 set R_1 out = 1
then R_4 set R_4 in = 1

2. Performing ALU op

eg $R_3 = R_1 + R_2$

S-1 : R_1's Output & Y's input are enables to transfer
S-2 : Multiplexer selects signals
S-3 : Contents of Z in R_3 as result

3. Fetching a word frm mmry

MFC (Memory func Completed) : The processor waits for an MFC signal to indicate the completion of a memory op

eg : MOV (R1), R_2

S-1 : load R_1's add to MAR
S-2 : Wait for MFC
S-3 : Then load data to MDR
S-4 : Transfer frm MDR to R_2

4. Storing a word to memory

eg : Move R_2, (R_1)

S-1 : load MAR to R_1's add
S-2 : load MDR with R_2's data & wait MFC

- **Branch Instruction**

Updates the PC with a new add, directing the flow of the the program

Offset x : The difference b/w the target add & add next Inst

eg : PC out, MAR in, Read, Add, Zin
Z out, PC in, Yin
IR out, Add, Zin
Zout, PC in, END

- Multiple Bus

3-bus setup : Uses 3 buses (A, B & C) for efficient data transfer

Regs : stores all Gen purpose regs with C input & A, B output

ALU : The ALU can directly pass one of its Inputs (A/B) as specified by control signals

PC : It inc by 4 bytes after loading

Component	I	O
PC	PCin	PCout
Register	R1in	R1out
ALU	A, B4	R=B
IR	IRin	IR out
MDR	MDRin	MDR out
MAR	MAR in	MAR out

Hardwired Control

- Control Unit Signals

These sig control steps of instruction execution in the processor

i) Microprogrammed Control :-
- uses a seq of binary codes to gen sig
- Adaptable but slower eg (CISC)

ii) Harwired Control

- They use dedicated circuits to gen sig
- finite state machine
- Fast but not flexible eg (RISC)

Here, Control sig are influenced by :-

1. Step Counter : track current step
2. IR : Holds active step
3. Condition flags : takes BRANCH decision

4. External sig : MFC type outside processor

- Each step completed in one clock cycle
- Counter keeps track of it

eg Add (R3), R1

PC out, MAR in, Read, Select 4, Add, Zin (inc PC 4↑, value stored)
Z out, PC in, Yin, WMFC
MDR out, IR in
R3 out, MAR in, Read
R1 out, Yin, WMFC
MDR out, Select Y, Add, Zin
Zout, R1in, END

Key Points

- PC: Holds the address of the next instruction.
- MAR: Holds the memory address for reading or writing data.
- MDR: Temporarily holds data read from or written to memory.
- IR: Holds the current instruction being executed.
- Y and Z: Temporary registers used for holding intermediate values.

- Logic function for Signals

1. Zin

- It triggers for ADD & BRANC instruc
- Active based on Control steps

2. END

- Marks completion of an instruc
- Triggered / Active in ADD & Unconditional branching

3. RUN

- Advances the counter each clock cycle when set to 0

- Micro-programmed Control

- Microprogrammed control generates control signals using a seq of bits similar to machine lang
- Each bit indicates specific sig eg PC out at 1 or Pin at 0

- Control Word

eg Add (R_3), R_1

- Here Every step in an instruc execution requires genrating a specific Control word (CW)
- each CW in this sequence is called a microinstruc
- Each CPU instruc has its own micro routine
- They are stored in the Control store
- Basic Organisation of Microprog
  - The microprog Counter (MPC) reads CWs from Control store
  - When new instruc is loaded into IR
  - Add generator provides starting add
  - MPC inc with each clock cycle
  - Basic microprog. lacks ability to check condition codes / external inputs.
  - These microinstruc can branch to diff add based on external condition
- Branching in Microinstruc
  1. Mircoinstruction can perform conditional jumps with microroutines
  2. Control units are improved to handle these condition and branch as needed
- Changes in starting add generator

  They consider : IR
  External Inputs
  Condition codes
- Reducing Microinstruc size

  Allocating 1 bit for each Control sig would make microinstruc long & inefficient

To optimize : Group mutually exclusive sig
Use binary coding

- **Pipelining**

It's a technique used to improve processor performance by allowing simultaneous execution of multiple instructions

Utilizes faster hardware
Arranging hardware so mul ops can occur simutaneously

Instruction pipelining

Here, multiple instruc executed in parallel
So a current instruc doesn't wait for prev to finish

2- stage pipeline

Cycle 1 : Fetch instruc I_1 & store in buffer B_1
Cycle 2 : " " I_2, while I_1 in " B_1

This cycle continues

Here there are 2 units : fetch & execution

4- stage pipeline

4 stages : 1. fetch
2. decode
3. execute
4. write

Cycle 1 : I_1 fetched
2 : I_2 fetched, I_1 decode
3 : I_3 " , I_2 " , I_1 executed
4 : I_4 " , I_3 " , I_2 " , I_1 written

- Pipeline performance

  - pipelining inc CPU instruc throughput
  - 4-stage pipelining is 4 times faster

Factors influencing performance :-

1. Cache mmry
2. Hazard detection
3. execution time

Benefits :-

1. Inc CPU throughput
2. reduces execution time

Challenges

1. Handling stalls due to cache misses
2. Real-world factors often reduce actual gains
3. Manage sync

- Hazards

- Hazards are situations that prevent the next instruc in a pipeline frm executing on time
- They reduce performance

- Types

1. Structural

Happens during a resource conflict coz h/w can't handle multiple instruc

2. Data

Occurs when curr step depends on result of prev step

3. Control

Arise from branch causing delays

- **Structural Hazards**

  - They occur due to conflict in resource
  - Single mmry for 2 instruc
  - Due to this pipeline waits

Load x(r1),r2

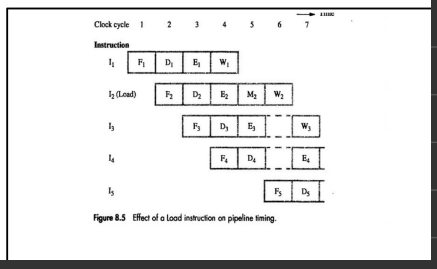

Figure 8.5 Effect of a Load instruction on pipeline timing.

- **Data Hazards**

Occurs when curr step depends on result of prev step

eg Add R_2, R_3, R_1 followed by SUB R_4, R_1, R_5

Here SUB needs result of ADD

Stalls are necessary if instruc need data that isn't ready

- Consider the following sequence of instructions:
- LD 0(R2),R1
- DSUB R4,R1,R5
- AND R6,R1,R7
- OR R8,R1,R9

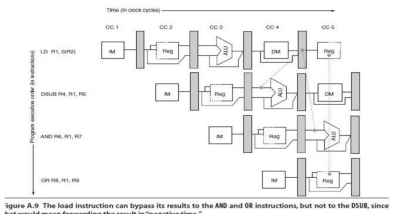

Figure A.9 The load instruction can bypass its results to the AND and OR instructions, but not to the DSUB, since that would mean forwarding the result in "negative time."

- Braches

- Unconditional Branch

- Branch instruc can disrupt pipeline by changing PC
- Branch penalty : It's the time lost due to branch instruction

- Instruction Queue

- processors use instruction queues to minimize stalls
- The fetch unit fills queue with instruc
- Due to this pipeline is activ

- Branch Prediction

  - It tries to guess whether a branch will be taken or not
  - eg. if one branch not taken, then it shld correct mistake to avoid delay

Types : 1. Static : decides before prog runs
2. Dyanamic : Changes based on prog's history

- Algorithms

A simple prediction has 2 stats

LT (likely to be Taken)
LNT (" not "

- 4 - state Algorithm

It's more advanced with 4 stats
1. ST : Strong likely to take
2. LT : "
3. LNT : " nor "
4. SNT : Strogly not "

eg Initially LNT
If taken then change to ST else SNT

- **Addressing Modes**

1. **Complex AM**

   May save prog but cause pipeline stalls & harder for compilers

Adv : reduces no. of instruc

disadv : Causes pipeline stall

2. **Simple AM**

   Faster & Easier to handle in a pipeline

- **Conditional Codes**

- Conditional code (flags) can limit the flexibility of instru reordering

- To allow more reordering, it's better if flags are affected by few instruc as possible