# COMP2121: Microprocessors and Interfacing

## Instruction Execution and Pipelining

http://www.cse.unsw.edu.au/~cs2121

Lecturer: Hui Wu

Term 2, 2019

1

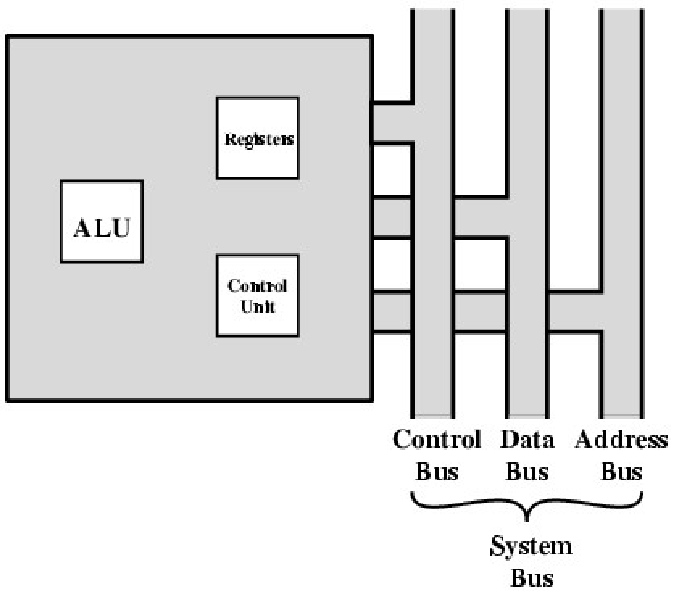
## Overview

- Processor organisation
- Instruction execution cycles
- Pipelining

2

2

## External View of Processor (1/4)



3

3

## External View of Processor (2/4)

- ALU
  - ❑ Performs arithmetic and logical operations (addition, subtraction, multiplication etc).
- Registers
  - ❑ General purpose registers
    - ❖ Used to stores temporary results.
  - ❑ Special purpose registers
    - ❖ Pointer registers, status register, program counter (PC) etc.
  - ❑ User-invisible registers
    - ❖ Used by the processor only. Typical user-invisible registers:
      - ➢ Memory buffer register (MBR)
      - ➢ Memory address register (MAR)
      - ➢ Instruction register (IR)

4

4

## External View of Processor (3/4)

- Control unit
  - ❑ Controls the flow of information through the processor, and coordinates the activities of other units within it.
  - ❑ Its functions vary with its internal architecture.
    - ❖ On a regular processor that executes x86 instructions natively, the control unit performs the tasks of fetching, decoding, managing execution and then storing results.
    - ❖ On a processor with a RISC core the control unit has significantly more work to do.

5

5

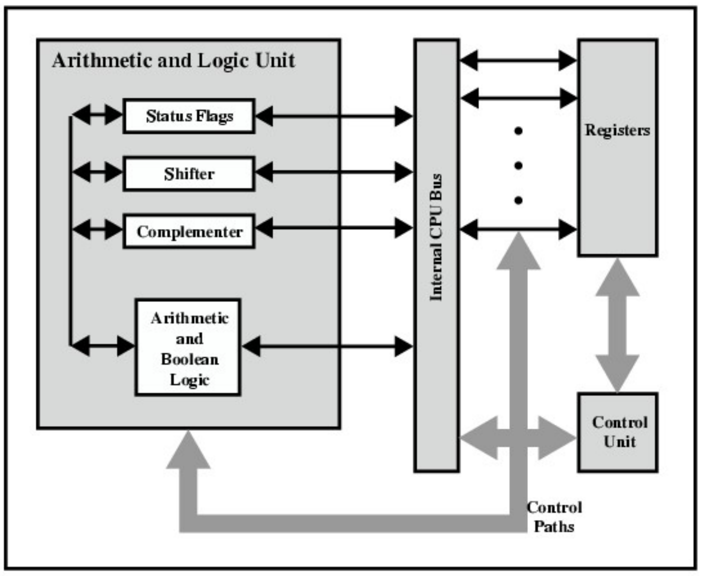
## External View of Processor (4/4)

- Buses
  - ❑ Data bus
    - ❖ Transfers data between the processor and other components (memory, I/O devices).
  - ❑ Address bus
    - ❖ Transfers the address from the processor to other components (memory, I/O devices).
  - ❑ Control bus
    - ❖ Transfers the control signals between the processor and other components (memory, I/O devices).
  - ❑ Details will be covered later.

6

6

## Internal View of Processor (1/2)



7

7

## Internal View of Processor (2/2)

- Status flags
  - ❑ Indicate the intermediate or final state or outcome of arithmetic and logical operations.
  - ❑ Example flags include V (2's complement oVerflow), S (Sign), Z (Zero) and C (Carry).
- Shifter
  - ❑ Performs shift operation.
- Complementer
  - ❑ Computes 2's complement.

8

8

## Register Organization

- space (temporary storage) for Processor
- User-visible registers
- User-invisible registers
- Control and status registers
- Number and function vary between processor designs
- One of the major design decisions
- Top level of memory hierarchy

## User Visible Registers

May be referenced by means of the machine instructions.

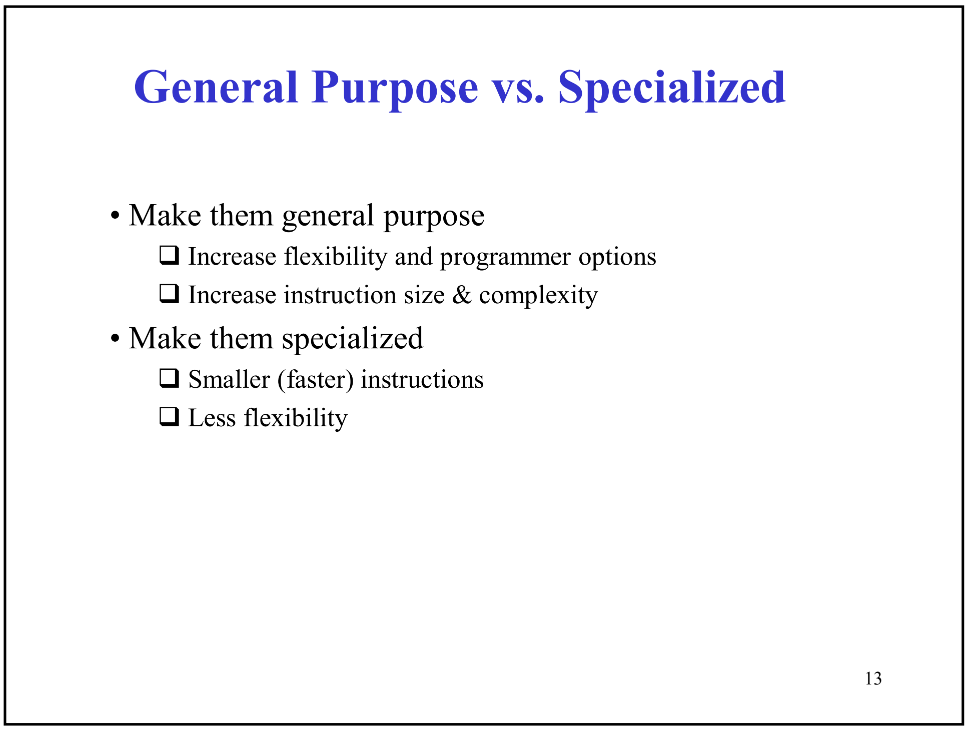
- General Purpose
- Data
- Address
- Condition Codes

## General Purpose Registers

- May be true general purpose (any general-purpose register can contain the operand for any opcode)
- May be restricted (registers for floating-point and stack operations)
- May be used for data or addressing
  - ❑ Data
    - ❖ Also called accumulator
    - ❖ r1~r31 in AVR.
  - ❑ Addressing
    - ❖ Segment registers

## Address Registers

- May be general purpose, or dedicated to a particular addressing mode.
- Segment pointers
  - ❑ CS and DS in Pentium processors.
- Index registers
  - ❑ X, Y and Z in AVR.
- Stack pointer
  - ❑ SP in AVR.

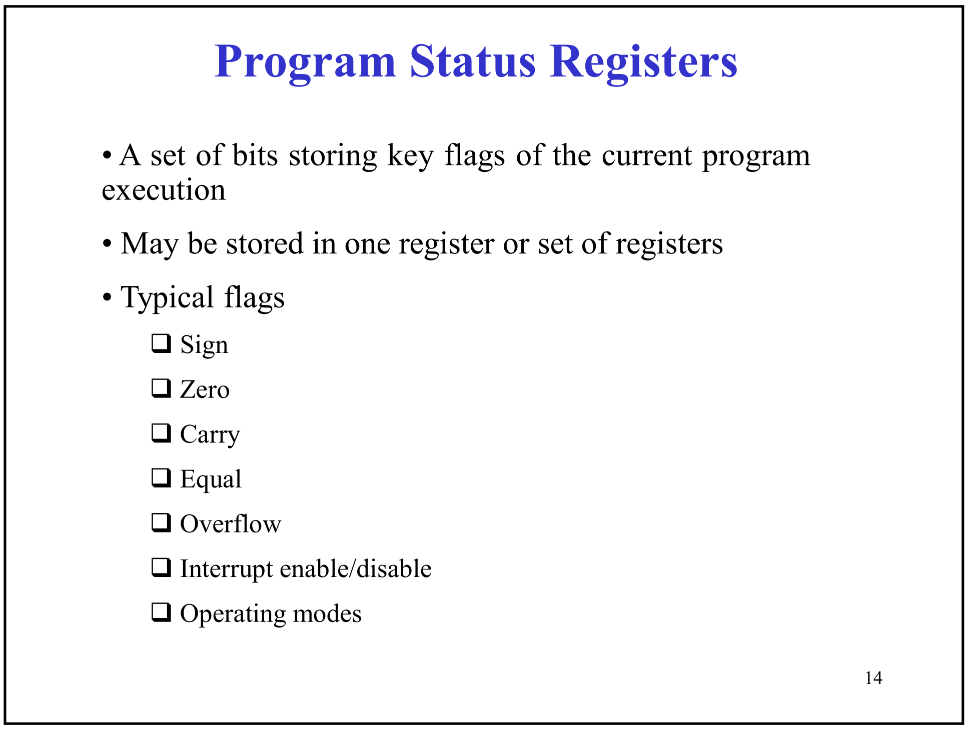


## General Purpose vs. Specialized

- Make them general purpose
  - ❑ Increase flexibility and programmer options
  - ❑ Increase instruction size & complexity
- Make them specialized
  - ❑ Smaller (faster) instructions
  - ❑ Less flexibility

13

13



## Program Status Registers

- A set of bits storing key flags of the current program execution
- May be stored in one register or set of registers
- Typical flags
  - ❑ Sign
  - ❑ Zero
  - ❑ Carry
  - ❑ Equal
  - ❑ Overflow
  - ❑ Interrupt enable/disable
  - ❑ Operating modes

14

14

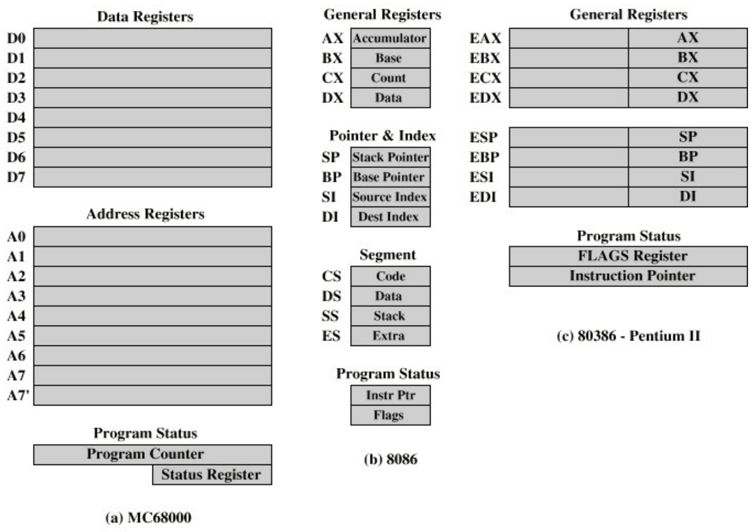
## Operating Modes

- Varies with processors.
- Typical operating modes:
  - ❑ Supervisor mode
    - ❖ Allows privileged instructions to execute
    - ❖ Used by operating system
    - ❖ Not available to user programs
  - ❑ User mode
    - ❖ Privileged instructions cannot be executed
    - ❖ Used by user program

15

15

## Example Register Organizations

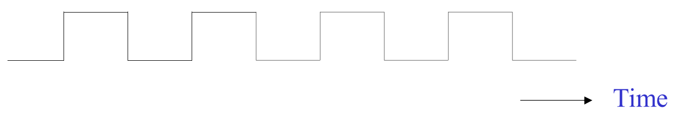


16

16

## Processor Cycle

- All modern processors are synchronous machines.
- Their timing is controlled by an external "clock" signal.
  - ❑ This is just a square electric pulse that is supplied to the processor (and memory etc) by an external source time.
  - ❑ A processor running at 1GHz receives $10^9$ clock pulses per second.
    - ❖ One pulse lasts 0.0000000001 second.



- The processor operations are therefore broken up in cycles.

17

17

## Instruction Cycle

- The instruction execution cycle is triggered by the clock cycle, but has several stages:
  - ❑ Each stage is triggered by successive clock pulses
  - ❑ The exact timing depends on the details of a particular processor
- A complete instruction cycle usually takes several clock cycles to execute.
- The instruction cycle is divided into several stages.
  - ❑ The number of stages vary with processors.

18

18

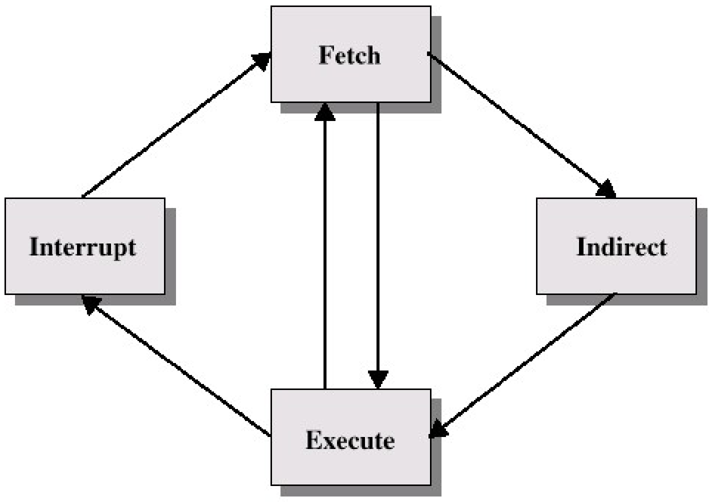
## Indirect Cycle

- May require memory access to fetch operands.
- Indirect addressing requires more memory accesses.
- Can be thought of as additional instruction subcycle.

19

19

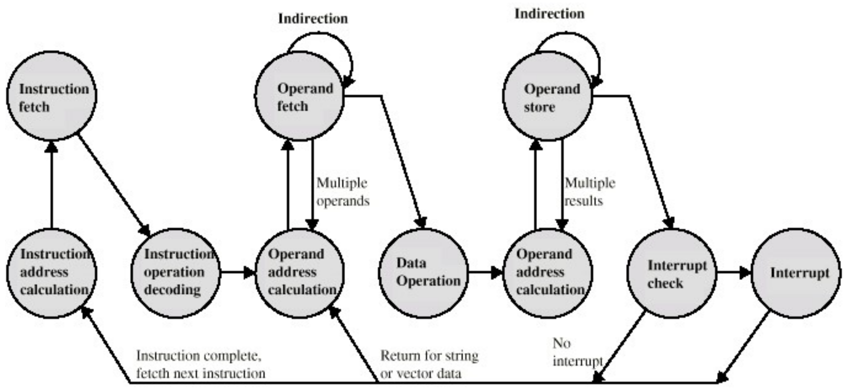
## Instruction Cycle with Indirect



20

20

## Instruction Cycle State Diagram


Indirection
Indirection
Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Interrupt check
Interrupt
Instruction complete, fetcth next instruction
Return for string or vector data
No interrupt

21

21

## Data Flow (Instruction Fetch)

- Depends on CPU design
- In general, Fetch:
  - ❑ PC contains address of next instruction
  - ❑ Address moved to MAR
  - ❑ Address placed on address bus
  - ❑ Control unit requests memory read
  - ❑ Result placed on data bus, copied to MBR, then to IR
  - ❑ Meanwhile PC incremented by 1

22

22

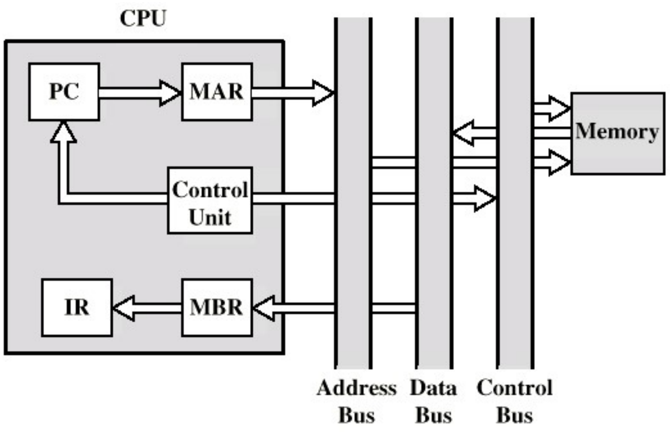
## Data Flow (Data Fetch)

- IR is examined.
- If indirect addressing, indirect cycle is performed.
  - ❑ Memory address is transferred to MAR.
  - ❑ Control unit requests memory read.
  - ❑ Result (address of operand) moved to MBR.

23

23

## Data Flow (Fetch Diagram)

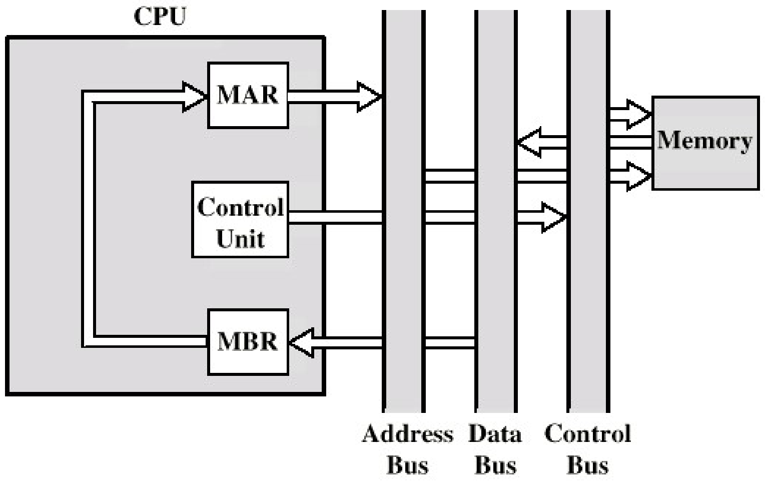


MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

24

24

## Data Flow (Indirect)



25

25

## Data Flow (Execute)

- May take many forms
- Depends on instruction being executed
- May include
  - ❑ Memory read/write
  - ❑ Input/Output
  - ❑ Register transfers
  - ❑ ALU operations

26

26

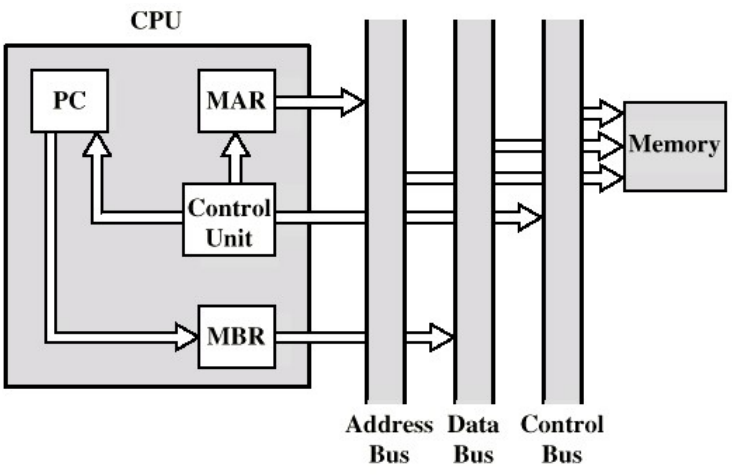
## Data Flow (Interrupt)

- Current PC saved to allow resumption after interrupt
- Contents of PC copied to MBR
- Special memory location (e.g. stack pointer) loaded to MAR
- MBR written to memory
- PC loaded with address of interrupt handling routine
- Next instruction (first of interrupt handler) can be fetched

27

27

## Data Flow (Interrupt)



28

28

## Instruction Pipelining

- Break the instruction cycle into stages
- Simultaneously work on each stage

29

29

## Instruction Prefetch

- Fetch accesses main memory
- Execution usually does not access main memory
- Can fetch next instruction during execution of current instruction
  - ❑ This is called instruction prefetch.

30

30

## Two Stage Instruction Pipeline

Break instruction cycle into two stages:

• FI: Fetch instruction

• EI: Execute instruction

| Clock cycle → | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Instruction i | FI | EI | | | | | |
| Instruction i+1 | | FI | EI | | | | |
| Instruction i+2 | | | FI | EI | | | |
| Instruction i+3 | | | | FI | EI | | |
| Instruction i+4 | | | | | FI | EI | |

## Two Stage Instruction Pipeline

• But not doubled:

❑ Fetch usually shorter than execution

❑ If execution involves memory accessing, the fetch stage has to wait

❑ Any jump or branch means that prefetched instructions are not the required instructions

• Add more stages to improve performance

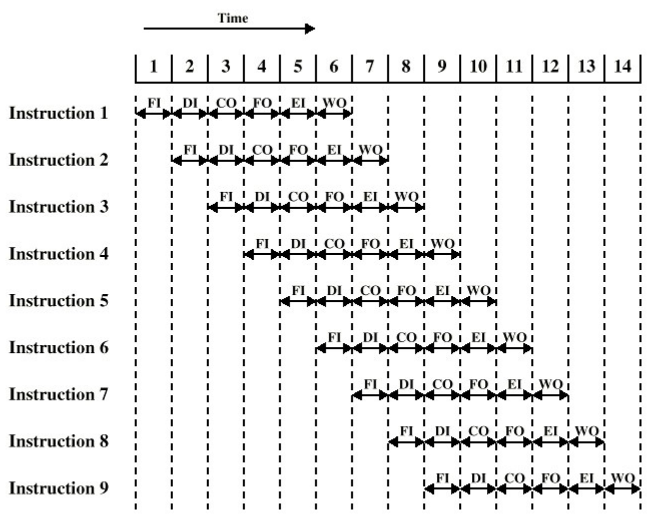
## Six Stage Pipelining

- Fetch instruction (FI)
- Decode instruction (DI)
- Calculate operands (CO)
- Fetch operands (FO)
- Execute instructions (EI)
- Write operand (WO)

33

33

## Timing for Six Stage Pipeline



34

34

## Theoretical Performance of Pipeline

- An ideal pipeline divides an instruction cycle into k stages.
  - ❑ Each stage requires 1 time unit
  - ❑ The instruction cycle requires k time units
- For n instructions, the execution times:
  - ❑ With no pipelining: nk time units
  - ❑ With pipelining: k + (n-1) time units
- Speedup of a k-stage pipeline is
  - ❑ $S = nk / [k+(n-1)] \approx k$ (for large n)

## The More Stages, the Better?

- The overhead in moving information between pipeline stages and synchronization between pipeline stages increases with the number of pipeline stages.
- Pipeline hazards make it difficult to keep a large pipeline at the maximum rate.

## Pipeline Hazards

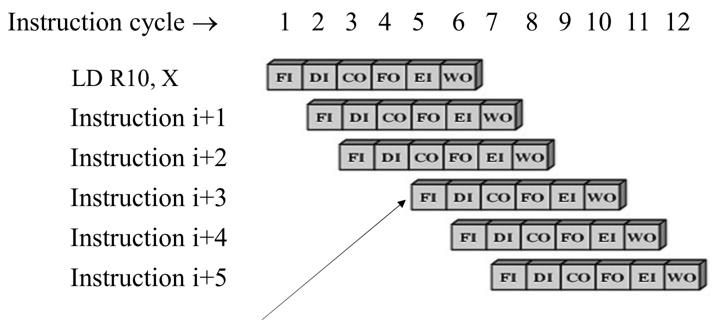
- Pipeline hazards are situations that prevent the next instruction in the instruction stream from executing during its designated clock cycles.
  - ❑ The instruction is said to be stalled.
  - ❑ If an instruction is stalled, all the following instructions are also installed.
- Types of pipeline hazards:
  - ❑ Structural hazards
  - ❑ Data hazards
  - ❑ Control hazards

37

37

## Structural Hazards

- A structural hazard occurs when multiple instructions need a resource ( e.g. memory) at the same time.



Instruction i+3 is stalled by one clock cycle

38

38

## Data Hazards

- A data hazard occurs when one instruction needs the result of another instruction, but the result is not available yet.

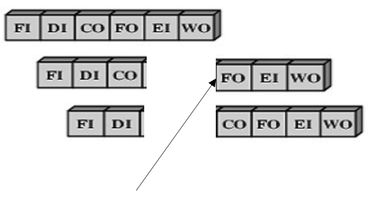
|  |  |
|---|---|
| MULT R2, R3 | R1:R0 ← R2*R3 |
| ADD R4, R0 | R4 ← R4+R0 |

Instruction cycle → 1 2 3 4 5 6 7 8 9 10 11

MULT R2, R3

ADD R4, R0

Instruction i+2



ADD R4, R0 is stalled by two clock cycles

39

## Control Hazards

- Control hazards are caused by branch instructions. Consider the following example:

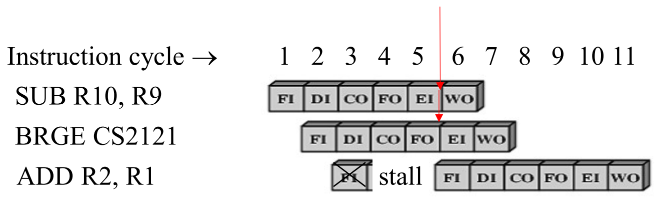
```
        SUB R10, R9
        BRGE CS2121
        SUB R12, R11
            •••
CS2121: ADD R2, R1
```

40

## Control Hazards (Cont.)

Case 1: Branch is taken.

At this moment, both the condition (set by SUB) and the target address are known. Since the branch is taken, ADD R2, R1 will be executed next. There is a penalty of 3 clock cycles in this case.



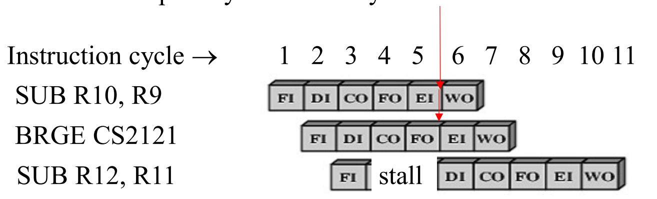
41

41

## Control Hazards (Cont.)

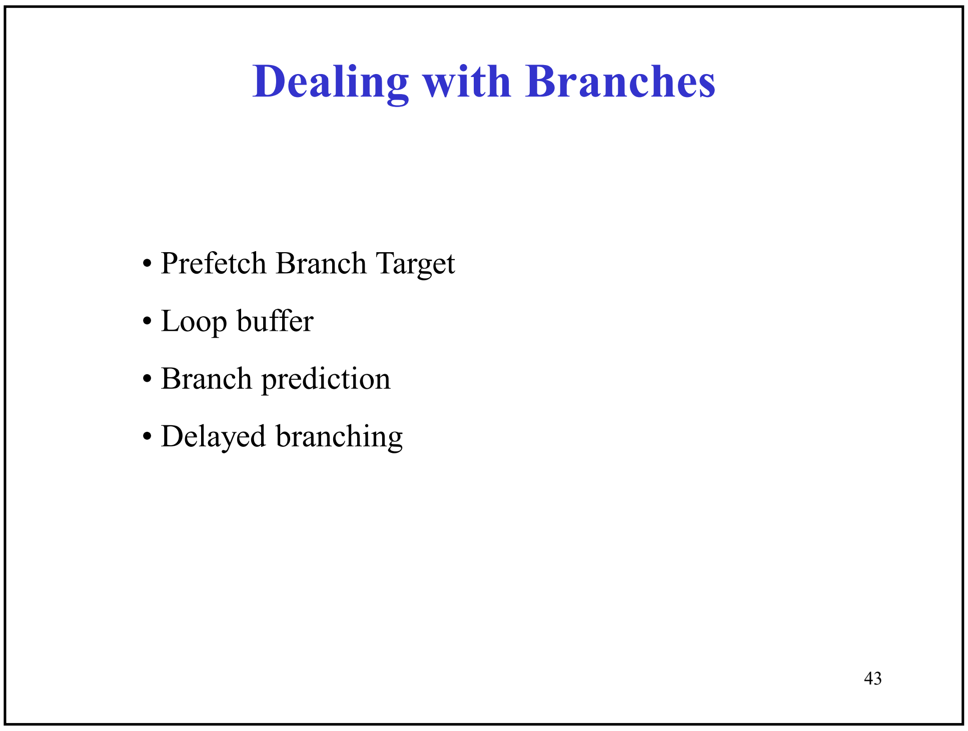
Case 2: Branch is NOT taken.

At this moment, both the condition (set by SUB) and the target address are known. Since the branch is not taken, SUB R12, R11 will be executed next. There is a penalty of 2 clock cycles in this case.



42

42

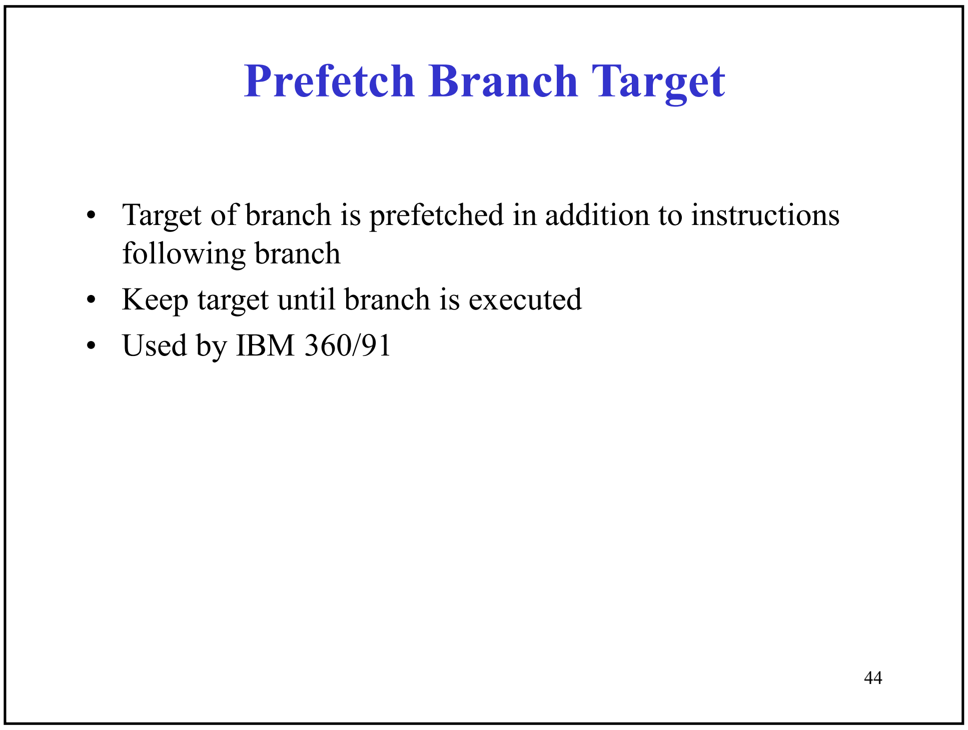


## Dealing with Branches

- Prefetch Branch Target
- Loop buffer
- Branch prediction
- Delayed branching

43

43



## Prefetch Branch Target

- Target of branch is prefetched in addition to instructions following branch
- Keep target until branch is executed
- Used by IBM 360/91

44

44

## Loop Buffer

- Very fast memory
- Maintained by fetch stage of pipeline
- Check buffer before fetching from memory
- Very good for small loops or jumps
- c.f. cache
- Used by CRAY-1

45

45

## Branch Prediction

- Static prediction
  - ❑ Predict never taken
    - ❖ Assume that jump will not happen
    - ❖ 68020 & VAX 11/780
  - ❑ Predict always taken
    - ❖ Assume that jump will happen

46

46

## Branch Prediction

- Dynamic prediction
  - ❑ One-bit prediction scheme
    - ❖ Used to record if the last execution resulted in a branch taken or not. The system predicts the same behavior as for the last time.
  - ❑ Two-bit prediction scheme
    - ❖ Used to record if the last two executions resulted in a branch taken or not.
  - ❑ Branch history table
    - ❖ Keep branch instruction address, history, target instruction (address) in a table in cache.
    - ❖ History info. can be used not only to predict the outcome of a conditional branch but also to avoid recalculation of the target address.

47

47

## Delayed Branch

- Do not take jump until you have to
- Rearrange instructions

Consider the following example:

```
        MULT R10, R9                          CP R4, R3
        MOVW R11:R10, R1:R0                   BRGE  CS2121
        CP R4, R3                             MULT R10, R9
        BRGE CS2121             ⟹             MOVW R11:R10
        ADD R8, R3                            ADD R8, R3
        •••                                   •••
CS2121: INC R10                       CS2121: INC R10
```

48

48

## Reading Material

1. Chapters 5&6. Computer Organization & Design: The HW/SW Interface by David Patterson and John Hennessy.