Subtraction

- Subtracting 6_{ten} from 7_{ten} directly

$$
\begin{array}{rl}
 & 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000111_{two} = 7_{ten} \\
 & 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000110_{two} = 6_{ten} \\
\hline
= & 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000001_{two} = 1_{ten}
\end{array}
$$

- Subtracting 6_{ten} from 7_{ten}using two's complement.

$$7 + (-6)$$

$$
\begin{array}{rl}
 & 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000111_{two} = 7_{ten} \\
+ & 11111111\ 11111111\ 11111111\ 11111111\ 11111111\ 11111111\ 11111111\ 11111010_{two} = -6_{ten} \\
\hline
= & 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000001_{two} = 1_{ten}
\end{array}
$$

Overflow

- Signed integers, addition
- When can an overflow occur?

Operand 1	Operand 2	Overflow	Check
+ve	-ve	No	
-ve	+ve	No	
+ve	+ve	Yes	-ve result
-ve	-ve	Yes	+ve result

Floating Point Numbers

Floating Point Numbers

- Convert (-11.75) base 10 to binary

Floating Point Numbers

- Convert (-11.75) base 10 to binary

11 to binary

11/2 = 5 remainder 1

5/2 = 2 remainder 1

2/2 = 1 remainder 0

1/2 = 0 remainder 1

11 to binary ⇒ 1011

Floating Point Numbers

- Convert (-11.75) base 10 to binary

11 to binary

11/2 = 5 remainder 1

5/2 = 2 remainder 1

2/2 = 1 remainder 0

1/2 = 0 remainder 1

11 to binary ➔ 1011

.75 to binary

.75 * 2 = 1. 50 integer part 1

.50 *2 = 1.00 integer part 1

Floating Point Numbers

- Convert (-11.75) base 10 to binary

11 to binary

11/2 = 5 remainder 1

5/2 = 2 remainder 1

2/2 = 1 remainder 0

1/2 = 0 remainder 1

11 to binary ➔ 1011

.75 to binary

.75 * 2 = 1. 50 integer part 1

.50 *2 = 1.00 integer part 1

$$\mathbf{11.75}\ \boldsymbol{in\ binary} \Rightarrow\ \mathbf{1011.11} \Rightarrow\ \mathbf{1011.11}\ *\ \mathbf{2^0}$$

In normalized scientific notation

$$\mathbf{1.01111}\ *\ \mathbf{2^3}$$

Floating Point Numbers

- Convert (-11.75) base 10 to binary

11 to binary

11/2 = 5 remainder 1

5/2 = 2 remainder 1

2/2 = 1 remainder 0

1/2 = 0 remainder 1

11 to binary ➔ 1011

.75 to binary

.75 * 2 = 1. 50 integer part 1

.50 *2 = 1.00 integer part 1

$$\mathbf{11.75}\ \boldsymbol{in\ binary} \Rightarrow\ \mathbf{1011.11} \Rightarrow\ \mathbf{1011.11}\ *\ \mathbf{2^0}$$

In normalized scientific notation

$$\mathbf{1.01111}\ *\ \mathbf{2^3}$$

S = 1 (sign bit)

Fraction = .01111

Exponent = 3 + bias (127) = 130 (in 8-bit binary is 1000 0010)

Floating Point Numbers

- Convert (-11.75) base 10 to binary

11 to binary

11/2 = 5 remainder 1

5/2 = 2 remainder 1

2/2 = 1 remainder 0

1/2 = 0 remainder 1

11 to binary ➔ 1011

.75 to binary

.75 * 2 = 1. 50 integer part 1

.50 *2 = 1.00 integer part 1

$$\mathbf{11.75}\ \boldsymbol{in\ binary} \Rightarrow\ \mathbf{1011.11} \Rightarrow\ \mathbf{1011.11}\ *\ \mathbf{2^0}$$

In normalized scientific notation

$$\mathbf{1.01111}\ *\ \mathbf{2^3}$$

S = 1 (sign bit)

Fraction = .01111

Exponent = 3 + bias (127) = 130 (in 8-bit binary is 1000 0010)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	0	0	0	0	1	0	0	1	1	1	1	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Floating-Point Example

- Represent –0.75
  - 0.75_{10} in binary:

    0.75 * 2 = 1.5 Integer part is 1

    0.5 * 2 = 1.0 Integer part is 1

    => $0.75_{10} = 0.11_2 = 1.1_2 \times 2^{-1}$
  - $-0.75 = (-1)^1 \times 1.1_2 \times 2^{-1}$

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

Double Precision

Exponent with bias used for representation (-1 + 1023 = **1022**)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 bit 11 bits 20 bits

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

32 bits

Convert Binary to decimal

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Convert Binary to decimal

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

– S = 1

Convert Binary to decimal

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- S = 1
- Exponent = 10000001_2 = $1*2^0+1*2^7=129$

Convert Binary to decimal

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- S = 1
- Exponent = $10000001_2 = 1*2^0+1*2^7=129$
- Fraction = $01000...00_2 = 1*2^{-2}$

Convert Binary to decimal

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- S = 1
- Exponent = $10000001_2 = 1*2^0+1*2^7=129$
- Fraction = $01000...00_2 = 1*2^{-2}$

- $x = (-1)^1 \times (1 + .01_2) \times 2^{(129 - 127)}$

  $= (-1) \times 1.25 \times 2^2$

  $= -5.0$

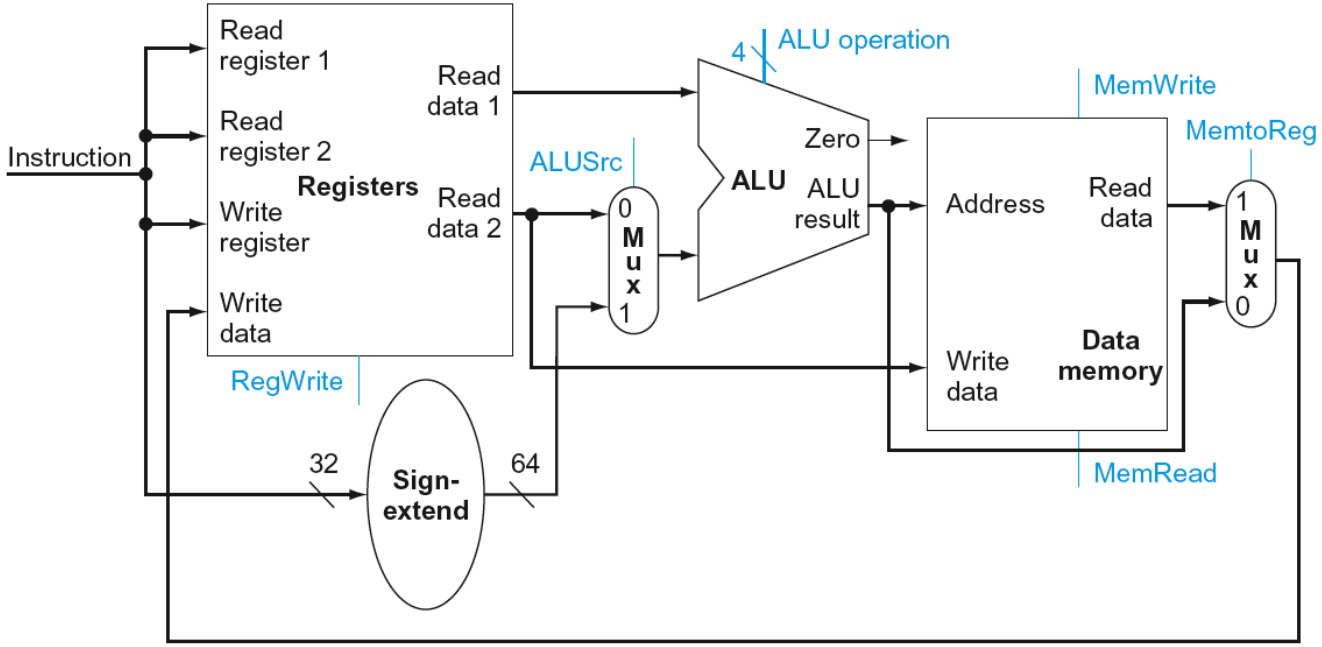
$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

Final Exam Review

- Chapter 1:
  - Performance
    - CPU Execution time
    - CPI
  - Amdahl's Law
- Chapter 2:
  - Number System
  - Load/Store data from/in memory
  - Assembly language
- Chapter 3:
  - Overflow
  - IEEE 754 representation
- Chapter 4:
  - Datapath
  - Pipelining
  - Hazards

Consider the following instruction:

Instruction: `AND Rd, Rn, Rm`

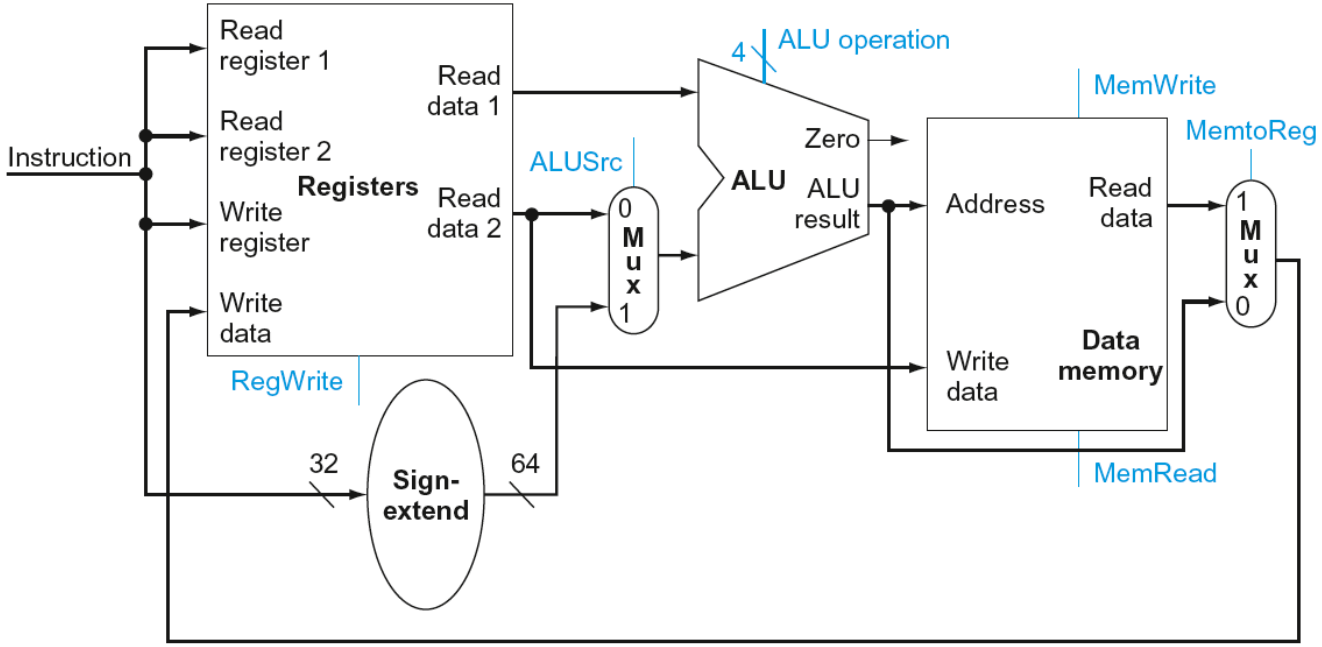


ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	pass input b
1100	NOR

1.1 What are the values of control signals

Consider the following instruction:

Instruction: `AND Rd, Rn, Rm`



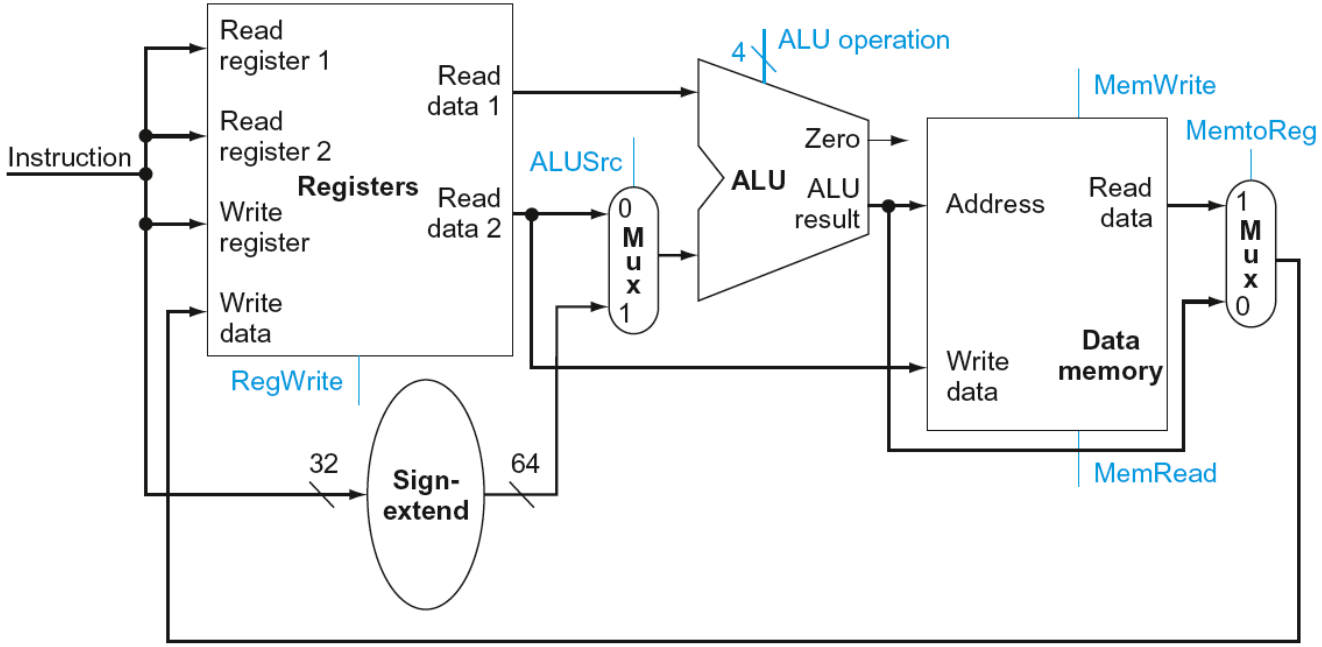
ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	pass input b
1100	NOR

1.1 What are the values of control signals

Signal	Value
RegWrite	
ALUSrc	
ALU operation	
MemWrite	
MemRead	
MemtoReg	

Consider the following instruction:

Instruction: `AND Rd, Rn, Rm`



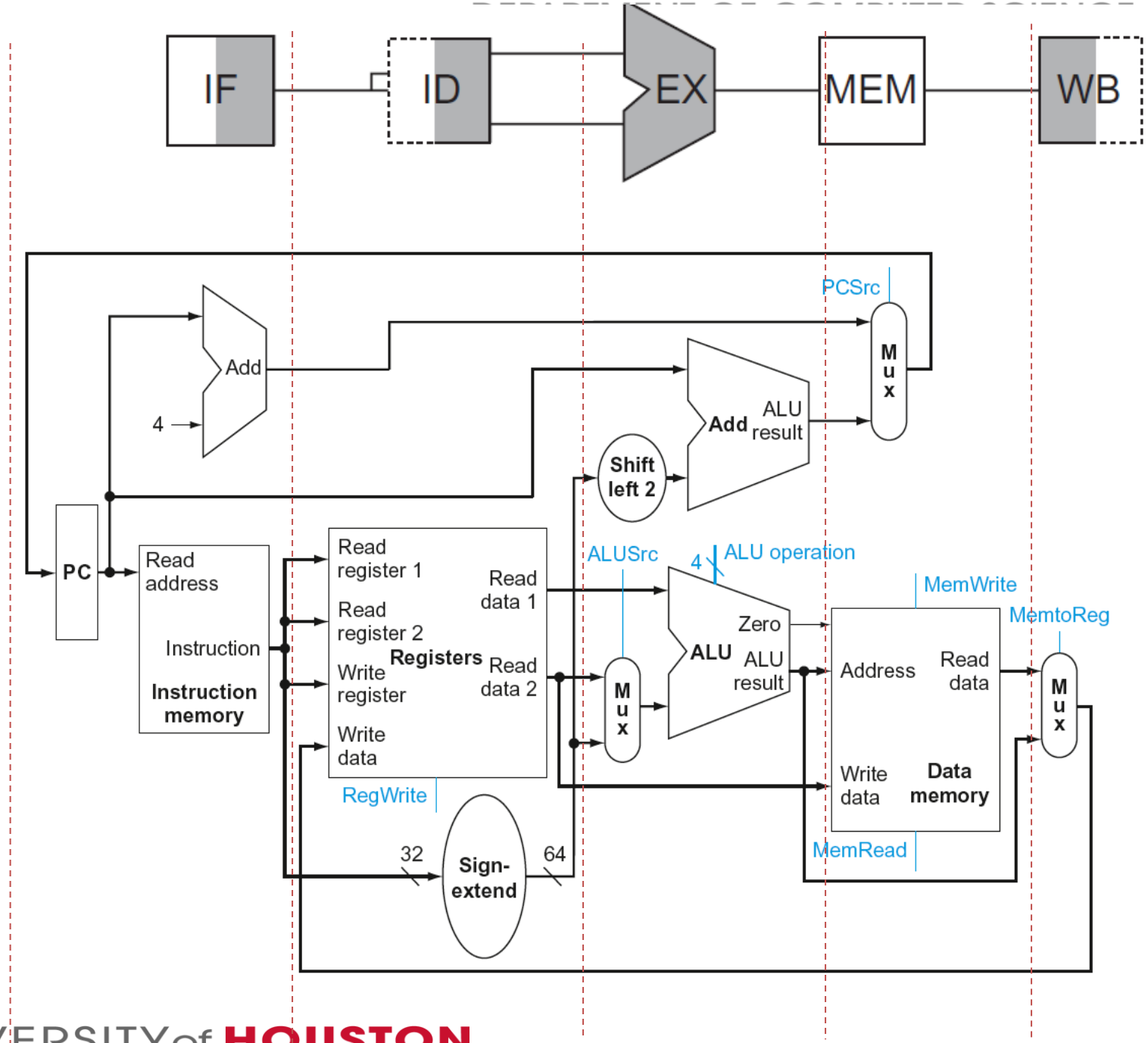
ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	pass input b
1100	NOR

1.1 What are the values of control signals

Signal	Value
RegWrite	1
ALUSrc	0
ALU operation	0000
MemWrite	0
MemRead	0
MemtoReg	0

Five stages, one step per stage

1. IF: Instruction fetch from memory
2. ID: Instruction decode & register read
3. EX: Execute operation or calculate address
4. MEM: Access memory operand
5. WB: Write result back to register



Pipeline Diagrams

- Show five-stage pipeline diagrams for a sequence of instructions.

Pipeline Diagrams

- Show five-stage pipeline diagrams for a sequence of instructions.

```
ADD X3, X1, X2
STUR X2, [X0,#24]
LDUR X4, [X0,#16]
```

Pipeline Diagrams

- Show five-stage pipeline diagrams for a sequence of instructions.

```
ADD X3, X1, X2
STUR X2, [X0,#24]
LDUR X4, [X0,#16]
```

Five stage pipeline

1. IF: Instruction fetch from memory
2. ID: Instruction decode & register read
3. EX: Execute operation or calculate address
4. MEM: Access memory operand
5. WB: Write result back to register

Pipeline Diagrams

- Show five-stage pipeline diagrams for a sequence of instructions.

ADD X3, X1, X2
STUR X2, [X0,#24]
LDUR X4, [X0,#16]

Inst./CC	1	2	3	4	5	6	7
ADD X3, X1, X2							
STUR X2, [X0,#24]							
LDUR X4, [X0,#16]							

Pipeline Diagrams

- Show five-stage pipeline diagrams for a sequence of instructions.

ADD X3, X1, X2
STUR X2, [X0,#24]
LDUR X4, [X0,#16]

Inst./CC	1	2	3	4	5	6	7
ADD X3, X1, X2	IF						
STUR X2, [X0,#24]							
LDUR X4, [X0,#16]							

Pipeline Diagrams

- Show five-stage pipeline diagrams for a sequence of instructions.

ADD X3, X1, X2
STUR X2, [X0,#24]
LDUR X4, [X0,#16]

Inst./CC	1	2	3	4	5	6	7
ADD X3, X1, X2	IF	ID					
STUR X2, [X0,#24]		IF					
LDUR X4, [X0,#16]							

Pipeline Diagrams

- Show five-stage pipeline diagrams for a sequence of instructions.

ADD X3, X1, X2
STUR X2, [X0,#24]
LDUR X4, [X0,#16]

Inst./CC	1	2	3	4	5	6	7
ADD X3, X1, X2	IF	ID	EXE				
STUR X2, [X0,#24]		IF	ID				
LDUR X4, [X0,#16]			IF				

Pipeline Diagrams

- Show five-stage pipeline diagrams for a sequence of instructions.

ADD X3, X1, X2
STUR X2, [X0,#24]
LDUR X4, [X0,#16]

Inst./CC	1	2	3	4	5	6	7
ADD X3, X1, X2	IF	ID	EXE	MEM	WB		
STUR X2, [X0,#24]		IF	ID	EXE	MEM	WB	
LDUR X4, [X0,#16]			IF	ID	EXE	MEM	WB

Pipeline Diagrams

- Show five-stage pipeline diagrams for a sequence of instructions.

Total cycle = 7

ADD X3, X1, X2
STUR X2, [X0,#24]
LDUR X4, [X0,#16]

Inst./CC	1	2	3	4	5	6	7
ADD X3, X1, X2	IF	ID	EXE	MEM	WB		
STUR X2, [X0,#24]		IF	ID	EXE	MEM	WB	
LDUR X4, [X0,#16]			IF	ID	EXE	MEM	WB

Hazards

Hazards

- Situations that prevent starting the next instruction in the next cycle

Hazards

- Situations that prevent starting the next instruction in the next cycle
- Structure hazards
  - Two instructions try to access the same resource simultaneously
- Data hazard
  - An instruction depends on completion of data access by a previous instruction
- Control hazard
  - Deciding on control action depends on previous instruction
  - Fetching next instruction depends on branch outcome
  - Pipeline can’t always fetch correct instruction

Data Hazards

- Three Generic Data Hazards
  - True Data Dependency: also called Read-After-Write (RAW)
  - Anti-dependency: also called Write-After-Read (WAR)
  - Output dependency: also called Write-After-Write (WAW)

Identifying Data Dependencies

List the true data dependencies

1. LDUR X1, [X0,#0]
2. LDUR X2, [X0,#8]
3. ADD X3, X1,X2
4. STUR X3, [X0,#24]
5. LDUR X4, [X0,#16]
6. ADD X5, X1,X4
7. STUR X5, [X0,#32]

Identifying Data Dependencies

List the true data dependencies

1. LDUR X1, [X0,#0]
2. LDUR X2, [X0,#8]
3. ADD X3, X1,X2
4. STUR X3, [X0,#24]
5. LDUR X4, [X0,#16]
6. ADD X5, X1,X4
7. STUR X5, [X0,#32]

Instruction 3 on 1
Instruction 3 on 2
Instruction 4 on 3
Instruction 6 on 1
Instruction 6 on 5
Instruction 7 on 6

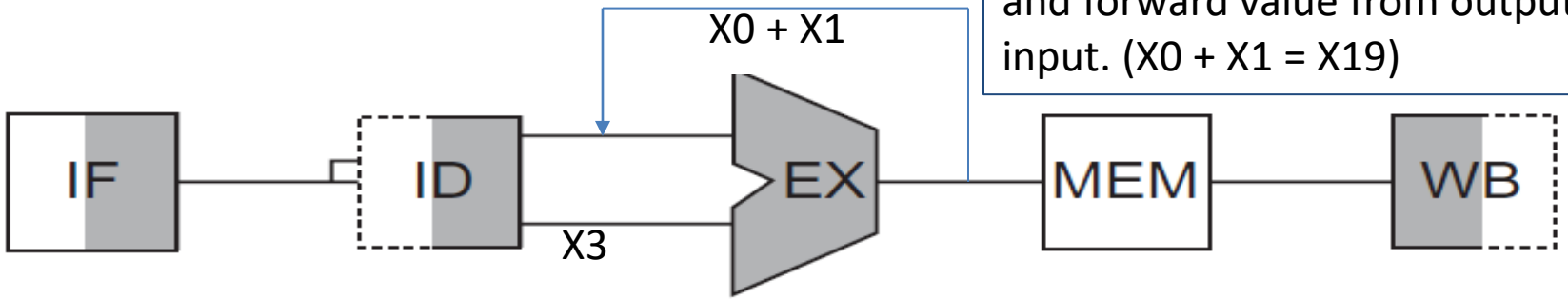
Overcoming Hazards

- Situations that prevent starting the next instruction in the next cycle
- Structure hazards
  - Not necessarily an issue in LEGv8
- Data hazard
  - Forwarding or Bypassing
- Control hazard
  - Deciding on control action depends on previous instruction
  - Fetching next instruction depends on branch outcome
  - Pipeline can’t always fetch correct instruction

Example R-Type Instruction

Cycle 4:
Inst 1 ➔ No Mem stage
Inst 2 ➔ ignore the loaded value, and forward value from output to input. (X0 + X1 = X19)

•



Cycles	1	2	3	4	5
ADD X19, X0, X1	Fetch instruction from mem	Read data from registers 0, 1	Add values of registers 0, 1 (value to write in reg 19 is computed)		
SUB X2, X19, X3		Fetch instruction from mem	Read data from registers 19, 3	Subtract X0 from X19	Forward value Bypass

Forwarding or Bypassing

Two cases:

1. Arithmetic instruction
2. Arithmetic instruction

Forwarding or Bypassing

Two cases:

1. Arithmetic instruction
2. Arithmetic instruction

1. Load instruction
2. Arithmetic instruction

Forwarding or Bypassing

Two cases:

1. Arithmetic instruction
2. Arithmetic instruction

1. Load instruction
2. Arithmetic instruction

Example:

```
ADD x19, x0, x1
SUB x2, x19, x3
```

Forwarding or Bypassing

Two cases:

1. Arithmetic instruction
2. Arithmetic instruction

Example:

```
ADD X19, X0,  X1
SUB X2,  X19, X3
```

1. Load instruction
2. Arithmetic instruction

```
LDUR X1 [ X2, #0]
SUB  X4, X1, X5
```

Forwarding or Bypassing

Two cases:

1. Arithmetic instruction
2. Arithmetic instruction

Example:

```
ADD X19, X0,  X1 (result X19
available after EXE stage)
SUB X2,  X19, X3 (Value of x19 needed
in exe stage)
```

1. Load instruction
2. Arithmetic instruction

```
LDUR X1 [ X2, #0]
SUB  X4, X1, X5
```

Forwarding or Bypassing

Two cases:

1. Arithmetic instruction
2. Arithmetic instruction

Example:

```
ADD X19, X0,  X1 (result X19
available after EXE stage)
SUB X2,  X19, X3 (Value of X19 needed
in exe stage)
```

1. Load instruction
2. Arithmetic instruction

```
LDUR X1 [ X2, #0] (result X1
available after MEM stage)
SUB  X4, X1, X5 (Value of X1 needed
in exe stage)
```

Forwarding or Bypassing

Two cases:

1. Arithmetic instruction
2. Arithmetic instruction

Example:

ADD X19, X0, X1 (result X19 available after EXE stage)
SUB X2, X19, X3 (Value of X19 needed in exe stage)

ADD X19, X0, X1	IF	ID	EXE	MEM	WB	
SUB X2, X19, X3		IF	ID	EXE	MEM	WB

1. Load instruction
2. Arithmetic instruction

LDUR X1 [X2, #0] (result X1 available after MEM stage)
SUB X4, X1, X5 (Value of X1 needed in exe stage)

Forwarding or Bypassing

Two cases:

1. Arithmetic instruction
2. Arithmetic instruction

Example:

ADD X19, X0, X1 (result X19 available after EXE stage)
SUB X2, X19, X3 (Value of X19 needed in exe stage)

ADD X19, X0, X1	IF	ID	EXE	MEM	WB	
SUB X2, X19, X3		IF	ID	EXE	MEM	WB

1. Load instruction
2. Arithmetic instruction

LDUR X1 [X2, #0] (result X1 available after MEM stage)
SUB X4, X1, X5 (Value of X1 needed in exe stage)

Forwarding or Bypassing

Two cases:

1. Arithmetic instruction
2. Arithmetic instruction

Example:

ADD X19, X0, X1 (result X19 available after EXE stage)
SUB X2, X19, X3 (Value of X19 needed in exe stage)

ADD X19, X0, X1	IF	ID	EXE	MEM	WB	
SUB X2, X19, X3		IF	ID	EXE	MEM	WB

1. Load instruction
2. Arithmetic instruction

LDUR X1,[X2, #0] (result X1 available after MEM stage)
SUB X4, X1, X5 (Value of X1 needed in exe stage)

LDUR X1, [X2, #0]	IF	ID	EXE	MEM	WB	
SUB X2, X1, X3		IF	ID	EXE	MEM	WB

Forwarding or Bypassing

Two cases:

1. Arithmetic instruction
2. Arithmetic instruction

Example:

ADD X19, X0, X1 (result X19 available after EXE stage)
SUB X2, X19, X3 (Value of X19 needed in exe stage)

ADD X19, X0, X1	IF	ID	EXE	MEM	WB	
SUB X2, X19, X3		IF	ID	EXE	MEM	WB

1. Load instruction
2. Arithmetic instruction

LDUR X1,[X2, #0] (result X1 available after MEM stage)
SUB X4, X1, X5 (Value of X1 needed in exe stage)

LDUR X1, [X2, #0]	IF	ID	EXE	MEM	WB	
SUB X2, X1, X3		IF	ID	EXE	MEM	WB

Forwarding or Bypassing

Two cases:

1. Arithmetic instruction
2. Arithmetic instruction

Example:

ADD X19, X0, X1 (result X19 available after EXE stage)
SUB X2, X19, X3 (Value of X19 needed in exe stage)

1. Load instruction
2. Arithmetic instruction

LDUR X1,[X2, #0] (result X1 available after MEM stage)
SUB X4, X1, X5 (Value of X1 needed in exe stage)

ADD X19, X0, X1	IF	ID	EXE	MEM	WB	
SUB X2, X19, X3		IF	ID	EXE	MEM	WB

LDUR X1, [X2, #0]	IF	ID	EXE	MEM	WB		
Stall							
SUB X2, X1, X3			IF	ID	EXE	MEM	WB

Forwarding or Bypassing

Two cases:

1. Arithmetic instruction
2. Arithmetic instruction

Example:

ADD X19, X0, X1 (result X19 available after EXE stage)
SUB X2, X19, X3 (Value of X19 needed in exe stage)

1. Load instruction
2. Arithmetic instruction

LDUR X1,[X2, #0] (result X1 available after MEM stage)
SUB X4, X1, X5 (Value of X1 needed in exe stage)

ADD X19, X0, X1	IF	ID	EXE	MEM	WB	
SUB X2, X19, X3		IF	ID	EXE	MEM	WB

LDUR X1, [X2, #0]	IF	ID	EXE	MEM	WB		
Stall							
SUB X2, X1, X3			IF	ID	EXE	MEM	WB

Pipeline diagram with forwarding and stalls

```
LDUR X1, [X0,#0]
LDUR X2, [X0,#8]
ADD X3, X1,X2
STUR X3, [X0,#24]
LDUR X4, [X0,#16]
ADD X5, X1,X4
STUR X5, [X0,#32]
```

Pipeline diagram with forwarding and stalls

Clock Cycles	1	2	3	4	5	6	7	8	9	10	11
LDUR X1, [X0,#0]	IF	ID	EXE	MEM	WB						
LDUR X2, [X0,#8]		IF	ID	EXE	MEM						
ADD X3, X1, X2			IF	ID	EXE						

Pipeline diagram with forwarding and stalls

Clock Cycles	1	2	3	4	5	6	7	8	9	10	11
LDUR X1, [X0,#0]	IF	ID	EXE	MEM	WB						
LDUR X2, [X0,#8]		IF	ID	EXE	MEM						
Stall											
ADD X3, X1, X2				IF	ID	EXE					

Pipeline diagram with forwarding and stalls

	1	2	3	4	5	6	7	8	9	10	11	12
LDUR X1, [X0,#0]	IF	ID	EXE	MEM	WB							
LDUR X2, [X0,#8]		IF	ID	EXE	MEM	WB						
Stall												
ADD X3, X1, X2				IF	ID	EXE	MEM	WB				
STUR X3, [X0,#24]					IF	ID	EXE	MEM	WB			

Pipeline diagram with forwarding and stalls

	1	2	3	4	5	6	7	8	9	10	11	12
LDUR X1, [X0,#0]	IF	ID	EXE	MEM	WB							
LDUR X2, [X0,#8]		IF	ID	EXE	MEM	WB						
Stall												
ADD X3, X1, X2				IF	ID	EXE	MEM	WB				
STUR X3, [X0,#24]					IF	ID	EXE	MEM	WB			
LDUR X4, [X0,#16]						IF	ID	EXE	MEM	WB		
ADD X5, X1, X4							IF	ID	EXE	MEM	WB	

Pipeline diagram with forwarding and stalls

Clock Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
LDUR X1, [X0,#0]	IF	ID	EXE	MEM	WB								
LDUR X2, [X0,#8]		IF	ID	EXE	MEM	WB							
Stall													
ADD X3, X1, X2				IF	ID	EXE	MEM	WB					
STUR X3, [X0,#24]					IF	ID	EXE	MEM	WB				
LDUR X4, [X0,#16]						IF	ID	EXE	MEM	WB			
Stall													
ADD X5, X1, X4								IF	ID	EXE	MEM	WB	
STUR X5, [X0,#32]									IF	ID	EXE	MEM	WB

Code Scheduling to Avoid Stalls

stall

stall

```
LDUR    x1, [x0,#0]
LDUR    x2, [x0,#8]
ADD     x3, x1, x2
STUR    x3, [x0,#24]
LDUR    x4, [x0,#16]
ADD     x5, x1, x4
STUR    x5, [x0,#32]
```

13 cycles

```
LDUR    x1, [x0,#0]
LDUR    x2, [x0,#8]
LDUR    x4, [x0,#16]
ADD     x3, x1, x2
STUR    x3, [x0,#24]
ADD     x5, x1, x4
STUR    x5, [x0,#32]
```

Code Scheduling to Avoid Stalls

Clock Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
LDUR X1, [X0,#0]	IF	ID	EXE	MEM	WB								
LDUR X2, [X0,#8]		IF	ID	EXE	MEM	WB							
LDUR X4, [X0,#16]			IF	ID	EXE	MEM	WB						
ADD X3, X1, X2				IF	ID	EXE	MEM	WB					
STUR X3, [X0,#24]					IF	ID	EXE	MEM	WB				
ADD X5, X1, X4						IF	ID	EXE	MEM	WB			
STUR X5, [X0,#32]							IF	ID	EXE	MEM	WB		

Total 11 Clock Cycles

Write First Then Read in Clock Cycle

ADD X0, X1, X2	IF	ID	EXE	MEM	WB				
Inst 2		IF	ID	EXE	MEM	WB			
Inst 3			IF	ID	EXE	MEM	WB		
ADD X4, X0, X1				IF	ID	EXE	MEM	WB	

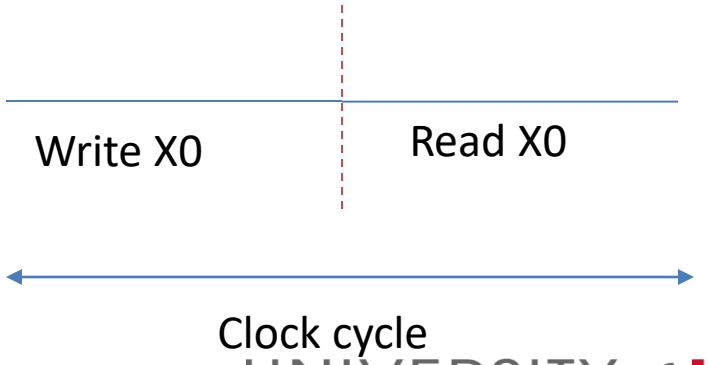
Write First Then Read in Clock Cycle

Value written in to X0

ADD X0, X1, X2	IF	ID	EXE	MEM	WB				
Inst 2		IF	ID	EXE	MEM	WB			
Inst 3			IF	ID	EXE	MEM	WB		
ADD X4, X0, X1				IF	ID	EXE	MEM	WB	

Value written in to X0

This is acceptable, the design is such that all writing happens in the first half of the clock cycle and reading in the second half.



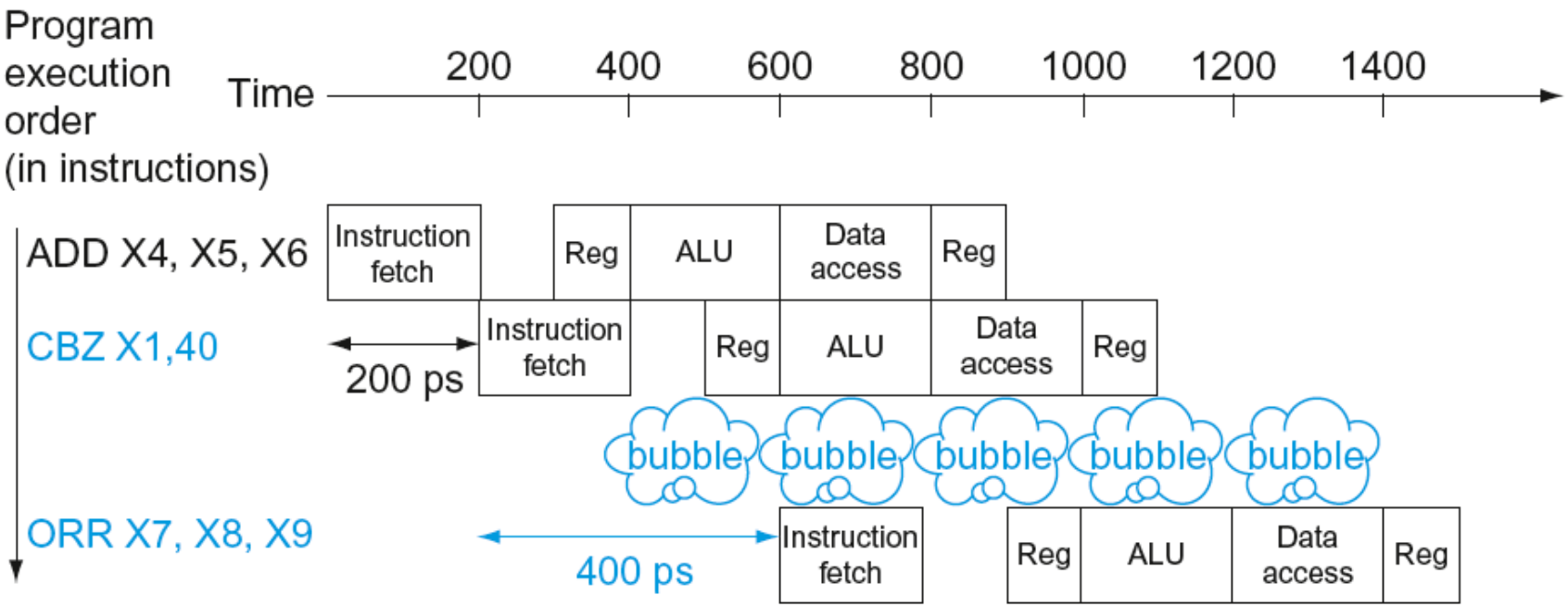
Control Hazards

- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipeline can't always fetch correct instruction
    - Still working on ID stage of branch

- In LEGv8 pipeline
  - Need to compare registers and compute target early in the pipeline
  - Add hardware to do it in ID stage

Stall on Branch

- Wait until branch outcome determined before fetching next instruction



Branch Prediction

- Longer pipelines can't readily determine branch outcome early
  - Stall penalty becomes unacceptable
- Predict outcome of branch
  - Only stall if prediction is wrong
- In LEGv8 pipeline
  - Can predict branches not taken
  - Fetch instruction after branch, with no delay

Branch prediction

Given the following code sequence in LEGv8 assembly language:

Assume that the variables i and k are associated with registers X1, and X3, respectively.

```
        ADDI X1, XZR, #1     ; i = 1
Loop:   SUBI X2, X1, #2
        CBZ  X2, Jump        ; Branch B1
        ADDI X3, X3, #1
Jump:   ADDI X1, X1, #1      ; i += 1
        SUBI X2, X1, #5
        CBZ  X2, Exit
        B Loop
Exit:
```

```
i = 1
while (i != 5) {
        if (i != 2 ){
                k = k + 1
        }
        i = i + 1
}
```

Value of i or X1	Actual outcome of branch b1
1	NT
2	T
3	NT
4	NT
5	NT

Strategy 1: Predict Always Taken

Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	1	T	NT	Yes
2	1	T	T	No
3	1	T	NT	Yes
4	1	T	NT	Yes
5	1	T	NT	Yes

Correct prediction: 1
Total predictions: 5

Accuracy:
1/5 = 0.2

1-bit predictor

- Predict the outcome of the next branch instruction to be the same as the previous outcome.
  - If previous outcome is taken, predict next as taken
  - If previous outcome is not taken then predict next as not taken.
- Flip the bit on incorrect prediction

Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	0	NT		
2				
3				
4				
5				

Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	0	NT	NT	no
2				
3				
4				
5				

Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	0	NT	NT	**no**
2	0	NT		
3				
4				
5				

Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	0	NT	NT	no
2	0	NT	T	yes
3				
4				
5				

Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	0	NT	NT	no
2	0	NT	T	yes
3	1	T		
4				
5				

Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	0	NT	NT	no
2	0	NT	T	yes
3	1	T	NT	yes
4				
5				

Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	0	NT	NT	no
2	0	NT	T	yes
3	1	T	NT	yes
4	0	NT		
5				

Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	0	NT	NT	no
2	0	NT	T	yes
3	1	T	NT	yes
4	0	NT	NT	no
5	0	NT	NT	no

Correct prediction: 3
Total predictions: 5

Accuracy:
3/5 = 0.6

Problem 2

Given the following code sequence in LEGv8 assembly language:

Assume that the variables i and k are associated with registers X1, and X3, respectively.

```
        ADDI X1, XZR, #1
Loop:   SUBI X2, X1, #2
        CBZ  X2, Jump        ; Branch B1
        ADDI X3, X3, #1
Jump:   ADDI X1, X1, #1
        SUBI X2, X1, #5
        CBZ  X2, Exit
        B Loop
Exit:
```

Show **branch prediction table** with 5 columns

1. Value of register X1
2. Branch predictor value for branch B1
   1. 0 ➔ not taken
   2. 1 ➔ taken
3. Prediction (T/NT)
4. Actual Outcome of B1
5. Miss prediction (yes, no)

Calculate the **prediction accuracy**

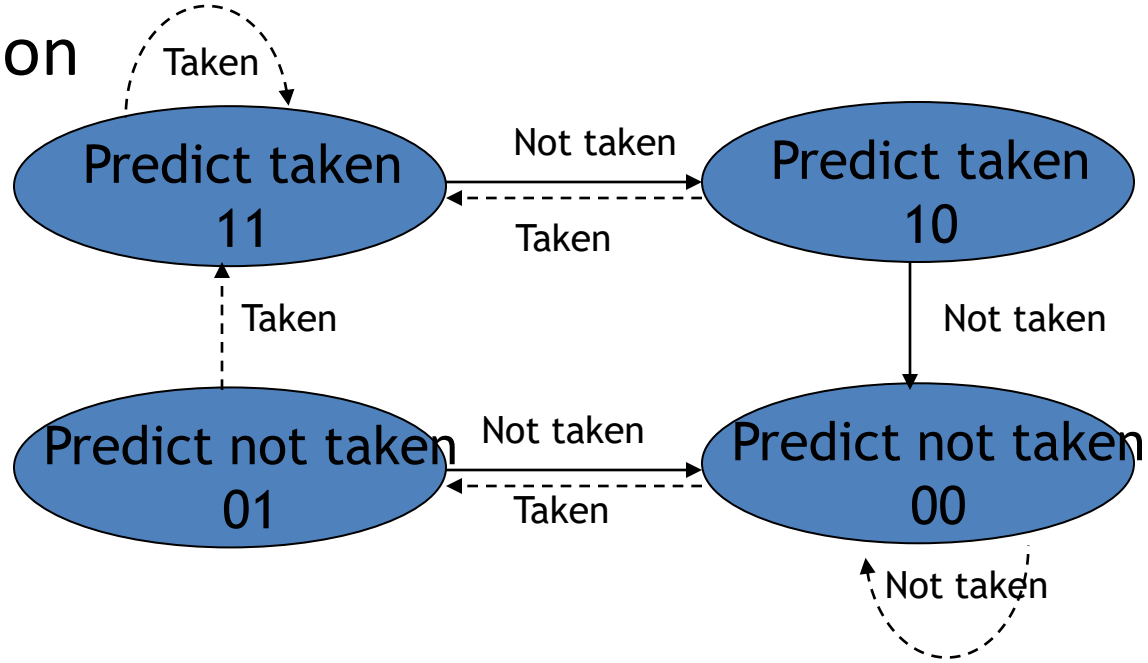
Branch Prediction Strategy:

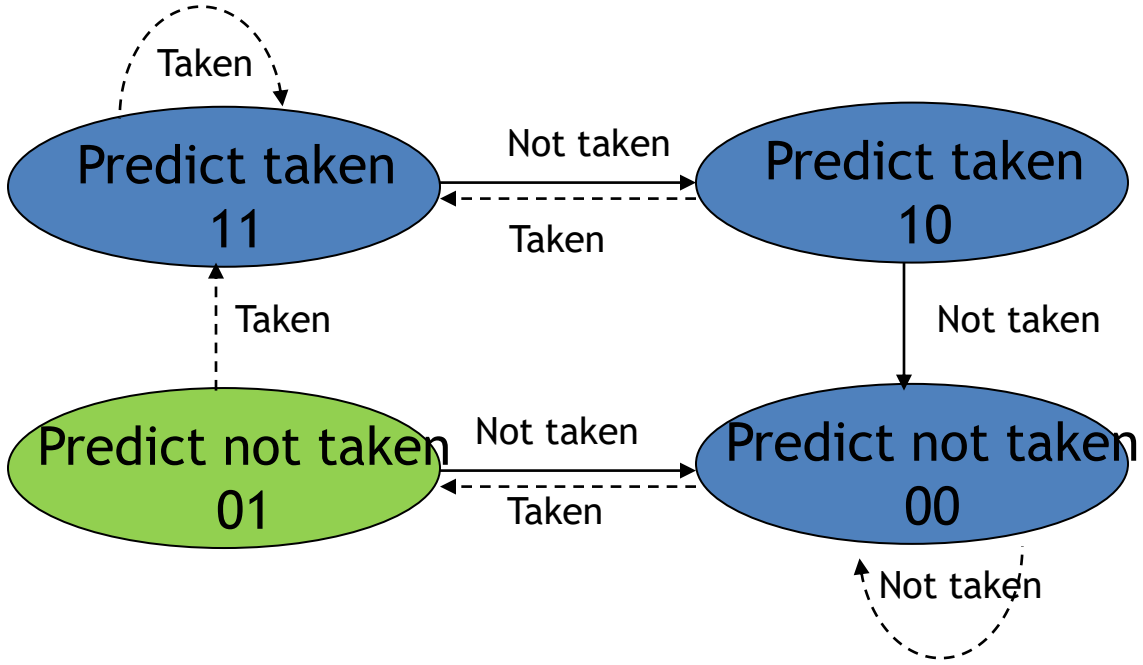
2-bit predictor

Initial state 01

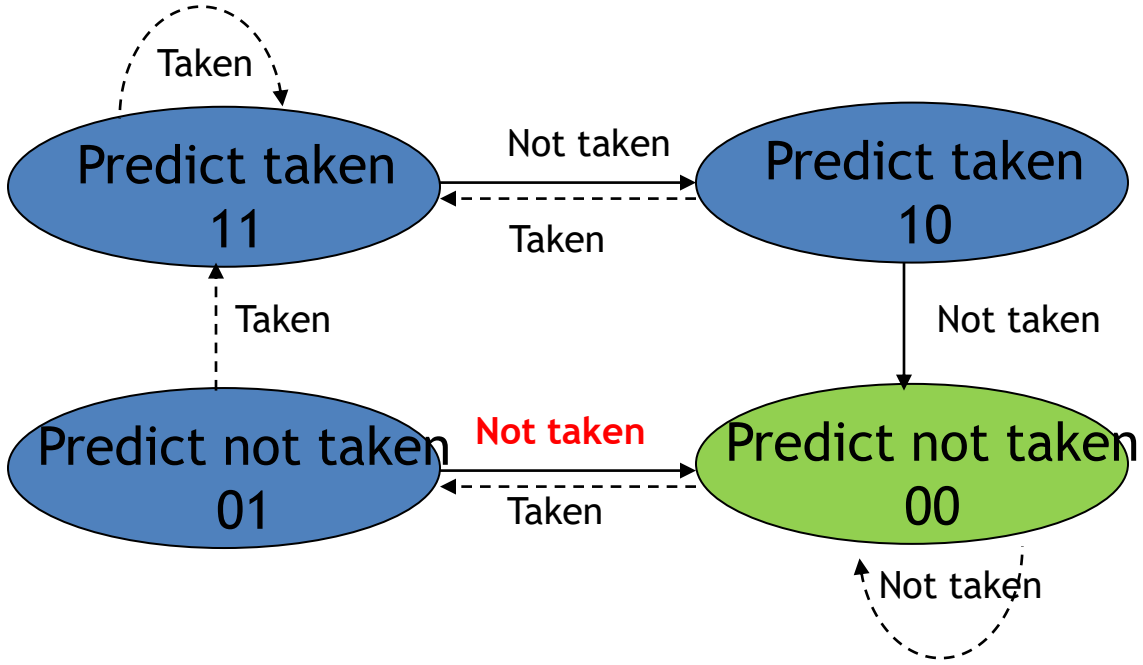
2bit Branch Prediction Buffer

- A prediction must miss twice before the prediction is changed
- Follow the State-Transition Diagram to update the predictor depending on the outcome of the branch instruction

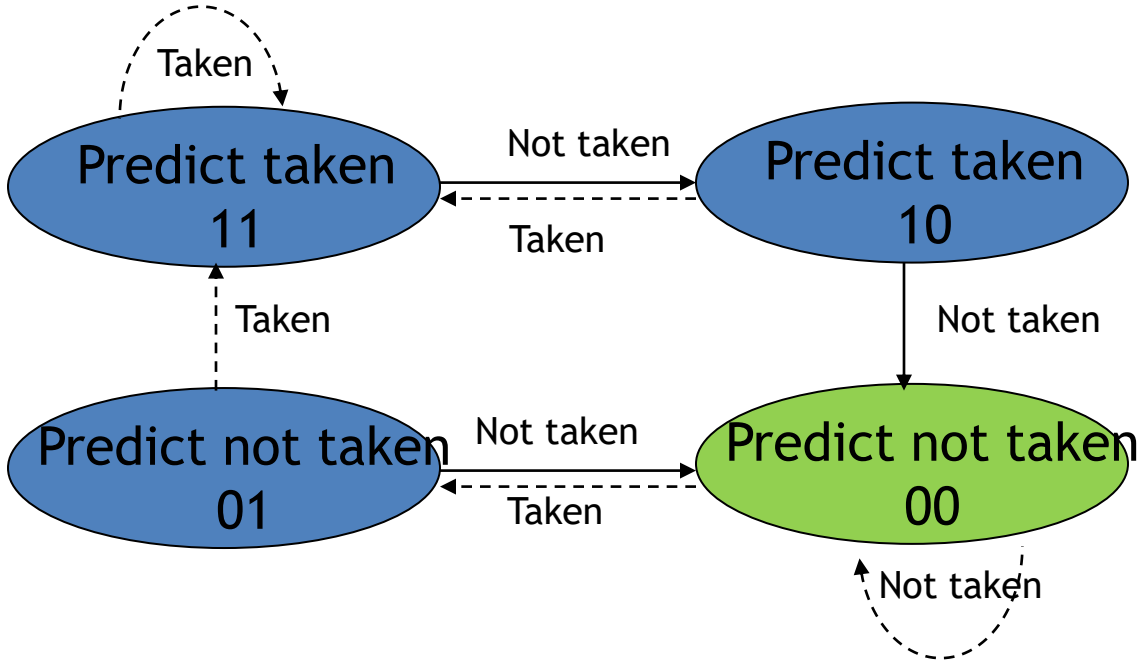




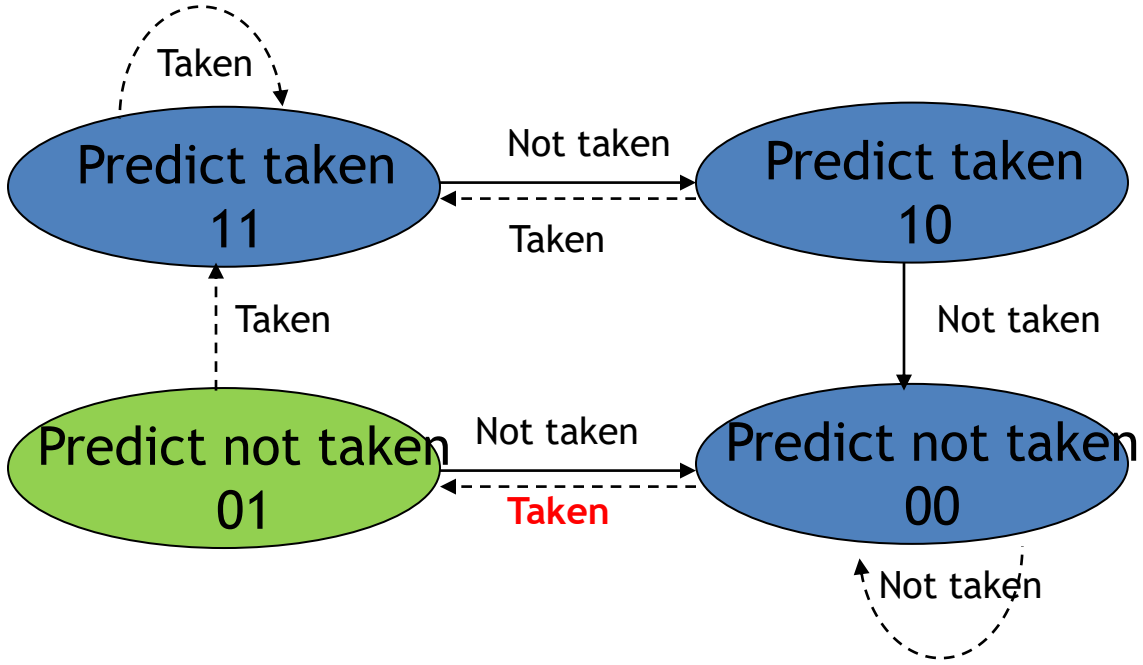
Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	01	NT		
2				
3				
4				
5				



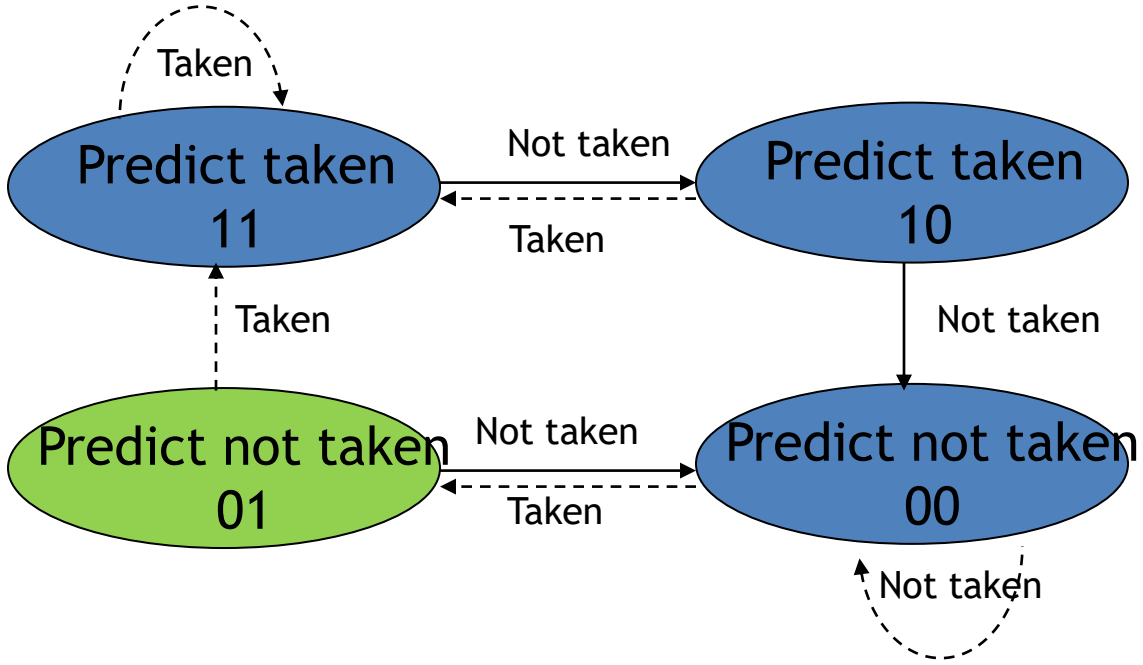
Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	01	NT	NT	no
2				
3				
4				
5				



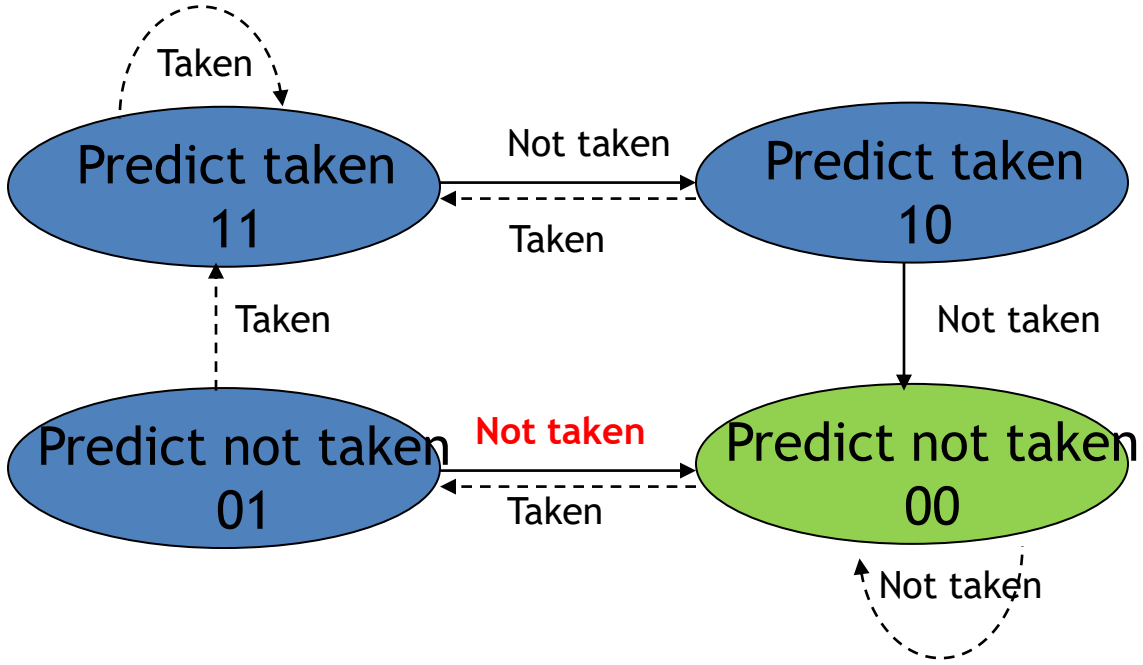
Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	01	NT	NT	no
2	00	NT		
3				
4				
5				



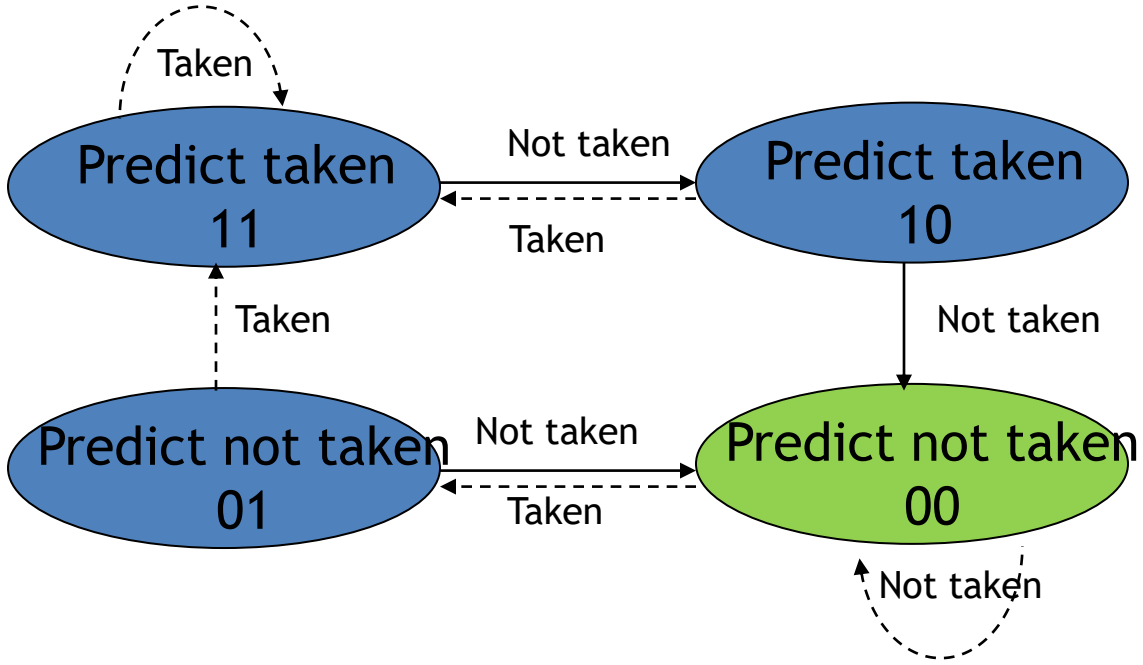
Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	01	NT	NT	no
2	00	NT	T	yes
3				
4				
5				



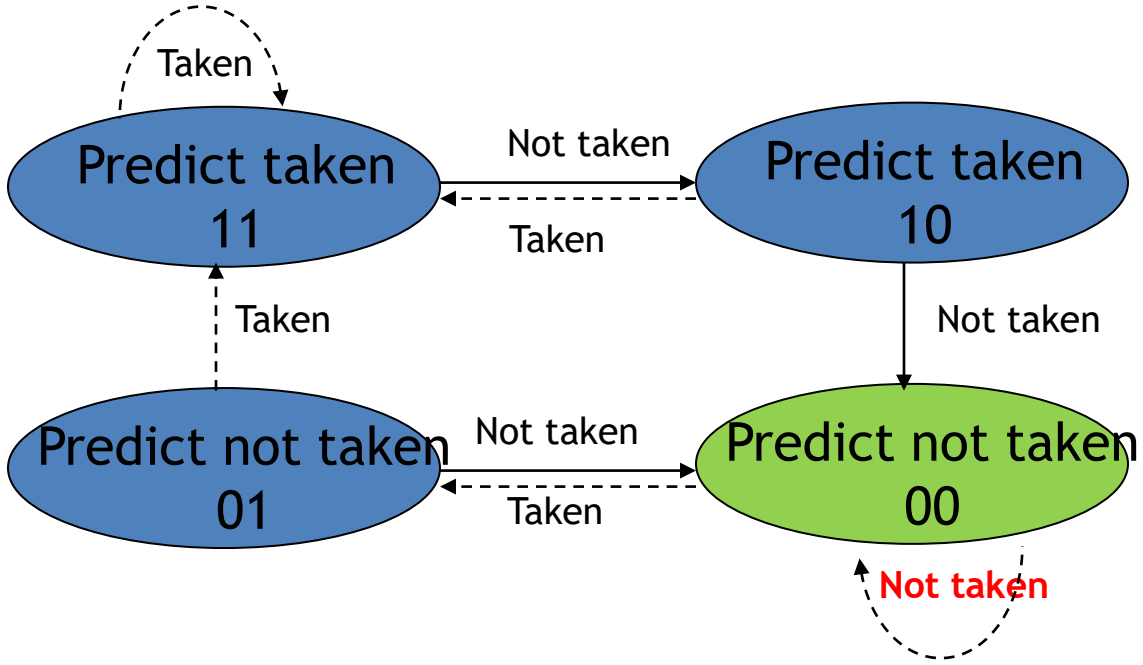
Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	01	NT	NT	no
2	00	NT	T	yes
3	01	NT		
4				
5				



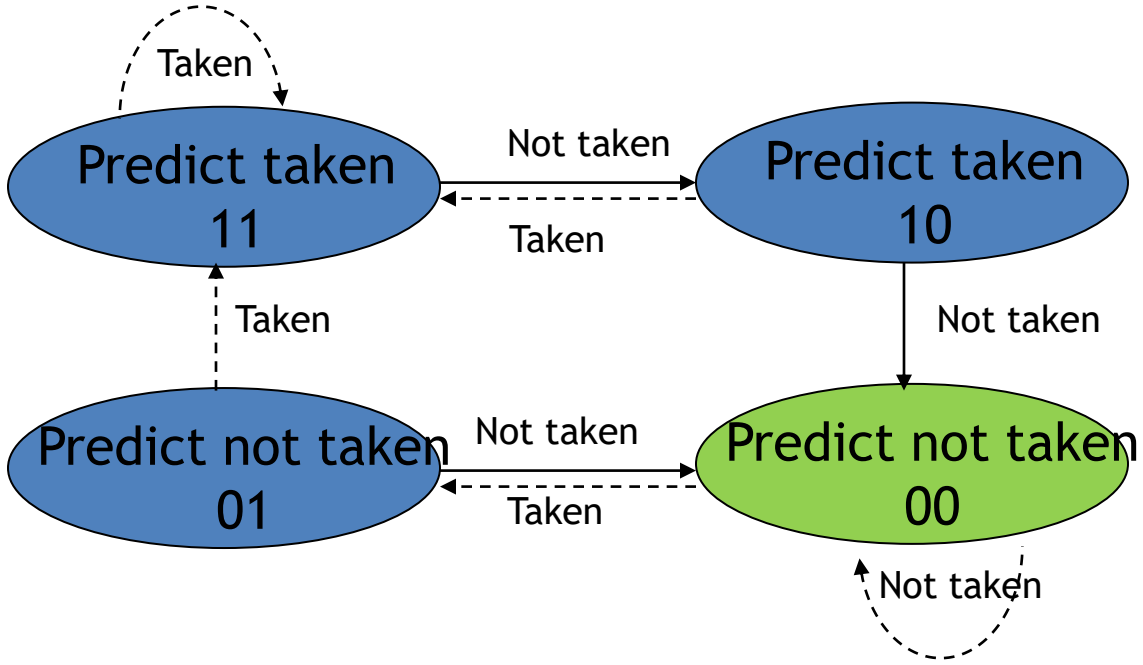
Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	01	NT	NT	no
2	00	NT	T	yes
3	01	NT	NT	no
4				
5				



Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	01	NT	NT	no
2	00	NT	T	yes
3	01	NT	NT	no
4	00	NT		
5				



Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	01	NT	NT	no
2	00	NT	T	yes
3	01	NT	NT	no
4	00	NT	NT	no
5				



Value of i or X1	Branch predictor for Branch b1	Prediction (T/NT)	Actual outcome of branch b1	Misprediction? (Yes/No)
1	01	NT	NT	no
2	00	NT	T	yes
3	01	NT	NT	no
4	00	NT	NT	no
5	00	NT	NT	no

Correct prediction: 4
Total predictions: 5

Accuracy:
4/5 = 0.8

Final Exam Review

- Chapter 1:
  - Performance
    - CPU Execution time
    - CPI
  - Amdahl's Law
- Chapter 2:
  - Number System
  - Load/Store data from/in memory
  - Assembly language
- Chapter 3:
  - Overflow
  - IEEE 754 representation
- Chapter 4:
  - Datapath
  - Pipelining
  - Hazards
- Chapter 5
  - Caches

Cache Organizations

- Types of Caches

1. Direct mapped cache
2. N-way set associative cache
3. Full associative cache

Direct Mapped Cache

- A main memory address is mapped to exactly one block in the cache.
- The mapping is determined by

Cache index = (Address) module (number of blocks in cache)

Example

- Cache with 16 Blocks.

Main memory location 10 is mapped to cache index _____.

index	Data
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Example

- Cache with 16 Blocks.

Main memory location 10 is mapped to cache index _____.

Number of blocks = 16

Cache index = (address) module (number of blocks)

index	Data
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Example

- Cache with 16 Blocks.

Main memory location 10 is mapped to cache index _____.

Number of blocks = 16

Cache index = (address) module (number of blocks)

Cache index = 10 module 16 = 10

index	Data
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Example

- Cache with 16 Blocks.

Main memory location 10 is mapped to cache index <u>10</u>.

Main memory location 2420 is mapped to cache index _____.

index	Data
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Example

- Cache with 16 Blocks.

Main memory location 10 is mapped to cache index <u>10</u>.

Main memory location 2420 is mapped to cache index _____.

2420 module 16 = 4

index	Data
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

N-Way Set Associative Cache

- A main memory address is mapped to a fixed number (n) of locations in the cache.
- The cache consist of sets each set has n-blocks. A main memory address is mapped to one set exactly, and with in the set the data can be placed in any block
- The set mapping is determined by

Set # = (Address) module (number of sets)

Example

- Total Cache Blocks 16
- 2-way set associative cache

2 blocks per set.

Total sets = 16/2

Example

- Total Cache Blocks 16
- 2-way set associative cache

2 blocks per set.

Total sets = 16/2 = 8

Set #	Data	
0		
1		
2		
3		
4		
5		
6		
7		

Example

- Total Cache Blocks 16
- 2-way set associative cache

2 blocks per set.

Total sets = 16/2 = 8

Main memory location 10 is mapped to set

_____

Set #	Data	
0		
1		
2		
3		
4		
5		
6		
7		

Example

- Total Cache Blocks 16
- 2-way set associative cache

2 blocks per set.

Total sets = 16/2 = 8

Main memory location 10 is mapped to set _____

Set # = (address) modulo (number of sets)

Set # = 10 module 8 = 2

Can be placed in either of the blocks in set 2

Set #	Data	
0		
1		
2		
3		
4		
5		
6		
7		

Example

- Total Cache Blocks 16
- 2-way set associative cache

2 blocks per set.

Total sets = 16/2 = 8

Main memory location 10 is mapped to set 2

Main memory location 2420 is mapped to set _____.

Set #	Data	
0		
1		
2		
3		
4		
5		
6		
7		

Example

- Total Cache Blocks 16
- 2-way set associative cache

2 blocks per set.

Total sets = 16/2 = 8

Main memory location 10 is mapped to set 2

Main memory location 2420 is mapped to set _____.

2420 modulo 8 = 4

Set #	Data	
0		
1		
2		
3		
4		
5		
6		
7		

Example

- Total Cache Blocks 16
- 4-way set associative cache

4 blocks per set.

Total sets = 16/4 = 4

Set #	Data			
0				
1				
2				
3				

Example

Set #	Data			
0				
1				
2				
3				

- Total Cache Blocks 16
- 4-way set associative cache

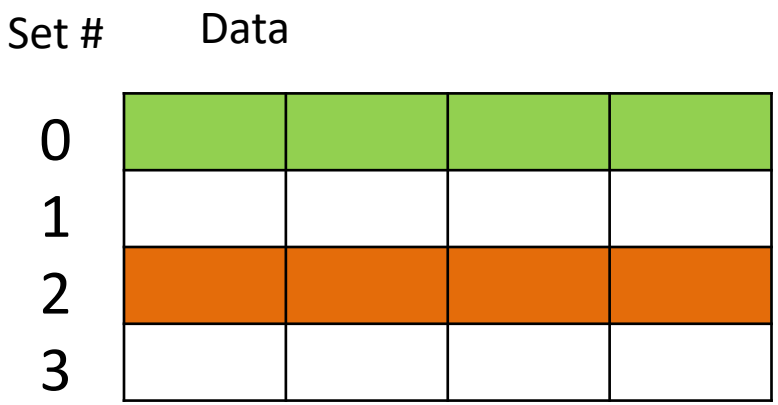
4 blocks per set.

Total sets = 16/4 = 4

Main memory location 10 is mapped to set _____

Main memory location 2420 is mapped to set _____.

Example



- Total Cache Blocks 16
- 4-way set associative cache

4 blocks per set.

Total sets = 16/4 = 4

Main memory location 10 is mapped to set 2

Main memory location 2420 is mapped to set 0.

Example

Set #	Data							
0								
1								

- Total Cache Blocks 16
- 8-way set associative cache

8 blocks per set.

Total sets = 16/8 = 2

Main memory location 10 is mapped to set 0

Main memory location 2420 is mapped to set 0.

Fully Associative Cache

- A main memory address can be mapped to any cache location

Example

- Cache with 16 Blocks.

Main memory location 10 is mapped to _____ cache location.

Data

Example

- Cache with 16 Blocks.

Main memory location 10 is mapped to <u>any</u> cache location.

Data

Cache Size

- 64 KB (64 * 1024 Bytes) of direct access cache
- Cache block size of 64 Bytes
- 48 bit addresses

Cache Size

- 64 KB (64 * 1024 Bytes) of direct access cache
- Cache block size of 64 Bytes
- 48 bit addresses

What is the byte offset?

Cache Size

- 64 KB (64 * 1024 Bytes) of direct access cache
- Cache block size of **64 Bytes**
- 48 bit addresses

What is the byte offset?

Use block size

64 = 2^6 ➔ 6 bits are used (least significant bits are used)

Cache Size

- 64 KB (64 * 1024 Bytes) of direct access cache
- Cache block size of **64 Bytes**
- 48 bit addresses

What is the byte offset? 6 bits

How many blocks are in the cache?

Cache Size

- 64 KB (64 * 1024 Bytes) of direct access cache
- Cache block size of **64 Bytes**
- 48 bit addresses

What is the byte offset? 6 bits

How many blocks are in the cache?

Blocks = (total size) / (Bytes per block) = $\frac{64*2^{10}}{64} = 2^{10}$

Cache Size

- 64 KB (64 * 1024 Bytes) of direct access cache
- Cache block size of **64 Bytes**
- 48 bit addresses

What is the byte offset? 6 bits

How many blocks are in the cache? 2^{10}

How many bits are used as cache index?

Cache Size

- 64 KB (64 * 1024 Bytes) of direct access cache
- Cache block size of **64 Bytes**
- 48 bit addresses

What is the byte offset? 6 bits

How many blocks are in the cache? 2^{10}

How many bits are used as cache index? 10 bits (least significant bits excluding the offset)

Cache Size

- 64 KB (64 * 1024 Bytes) of direct access cache
- Cache block size of **64 Bytes**
- 48 bit addresses

What is the byte offset? 6 bits

How many blocks are in the cache? 2^{10}

How many bits are used as cache index? 10 bits

How many bits are used for the tag?

Cache Size

- 64 KB (64 * 1024 Bytes) of direct access cache
- Cache block size of **64 Bytes**
- 48 bit addresses

What is the byte offset? 6 bits

How many blocks are in the cache? 2^{10}

How many bits are used as cache index? 10 bits

How many bits are used for the tag?

48 – 10 – 6 = 32

Cache Size

- 64 KB (64 * 1024 Bytes) of direct access cache
- Cache block size of **64 Bytes**
- 48 bit addresses

What is the byte offset? 6 bits

How many blocks are in the cache? 2^{10}

How many bits are used as cache index? 10 bits

How many bits are used for the tag? 32 bits

48 bit address 0100 1101 1001 0000 0010 0100 1101 1001 0000 0010 0100 1101

- What is the byte offset?
- What is the cache index to which this address will be mapped to?

48 bit address

0100 1101 1001 0000 0010 0100 1101 1001 0000 0010 0100 1101

tag

Cache index

Byte offset

- What is the byte offset?
- What is the cache index to which this address will be mapped to?

48 bit address 0100 1101 1001 0000 0010 0100 1101 1001 0000 0010 0100 1101

tag | Cache index | Byte offset

- What is the byte offset?

001101 ➔ 13

- What is the cache index to which this address will be mapped to?

48 bit address 0100 1101 1001 0000 0010 0100 1101 1001 0000 0010 0100 1101

tag | Cache index | Byte offset

- What is the byte offset?

001101 ➔ 13

- What is the cache index to which this address will be mapped to?

0000001001 ➔ 9

Cache Performance

Cache Miss/Hit Rate

Given the following code sequence calculating a matrix norm.

```
double a[96];
for (i=0; i<96; i=++ ) {
    a[i] = a[i]*a[i];
}
```

Assume a 64 KiB direct-mapped cache with a 32-byte block. What is the miss rate for the address stream above.

```
double a[96];
for (i=0; i< 96; i=++ ) {
    a[i] = a[i]*a[i];
}
```

- 64-bit representation
  - 32-byte blocks
  - 64-bits = 8 bytes per value
  - 32/8 = 4 values per block

```
double a[96];
for (i=0; i<100; i=++ ) {
    a[i] = a[i]*a[i];
}
```

Iteration	Hit/Miss
a[0]	Miss
A[1]	Hit
A[2]	Hit
A[3]	Hit
A[4]	Miss

1 Miss every 4 accesses

A[0]	A[1]	A[2]	A[3]
A[4]	A[5]	A[6]	A[7]

```
double a[96];
for (i=0; i<100; i=++ ) {
    a[i] = a[i]*a[i];
}
```

A[0]	A[1]	A[2]	A[3]
A[4]	A[5]	A[6]	A[7]

Miss rate = 1/4 = 0.25
Hit rate = 1 – miss rate = 0.75

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction