Lecture #7a

The Processor: Datapath

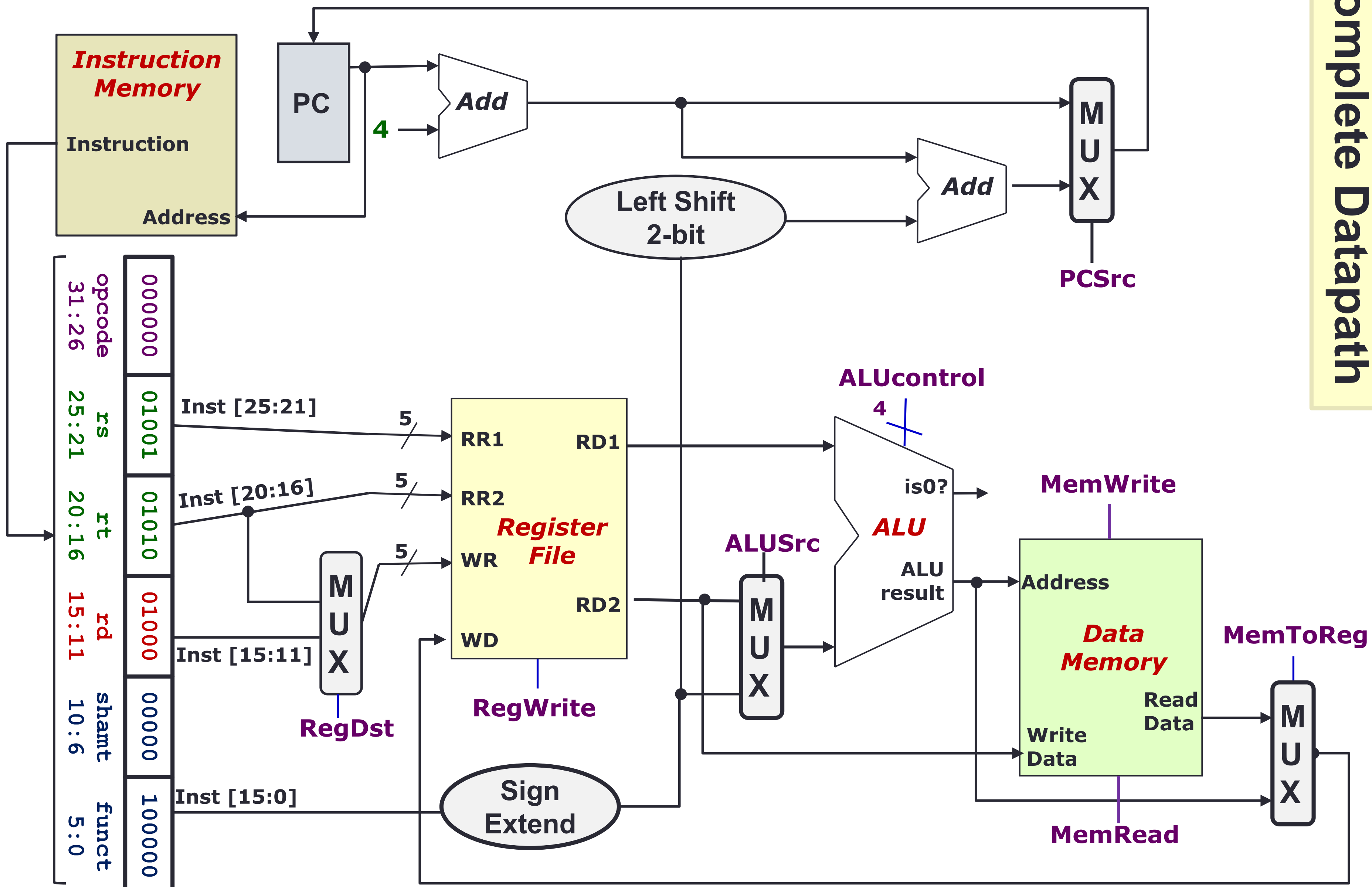
NUS National University of Singapore | School of Computing

Lecture #7a: Processor: Datapath

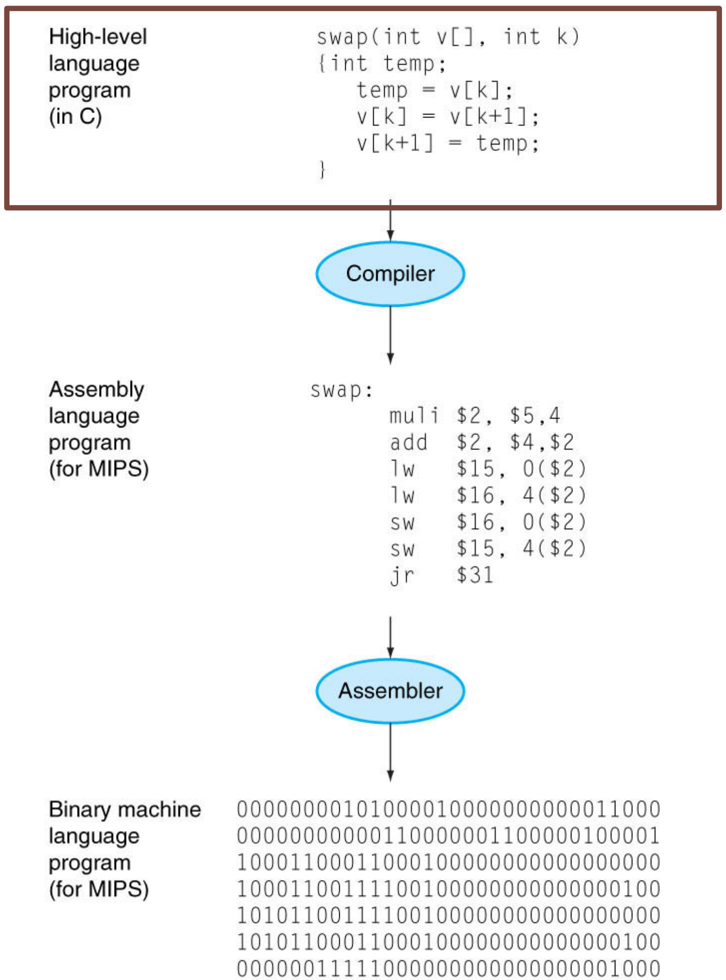
1. The Complete Datapath!
2. Brief Recap
3. From C to Execution
   - 3.1 Writing C Program
   - 3.2 Compiling to MIPS
   - 3.3 Assembling to Binaries
   - 3.4 Execution (Datapath)

Taken from CS2100 Lecture 11a.

Complete Datapath



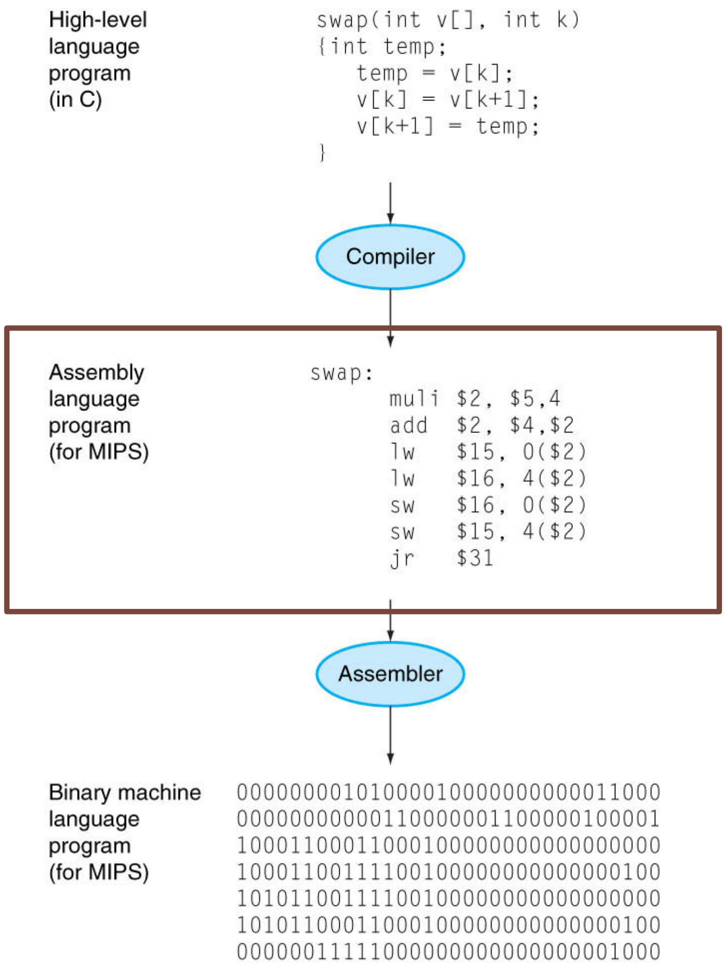
2. Brief Recap



- Write program in high-level language (e.g., **C**)

```
if(x != 0) {
  a[0] = a[1] + x;
}
```

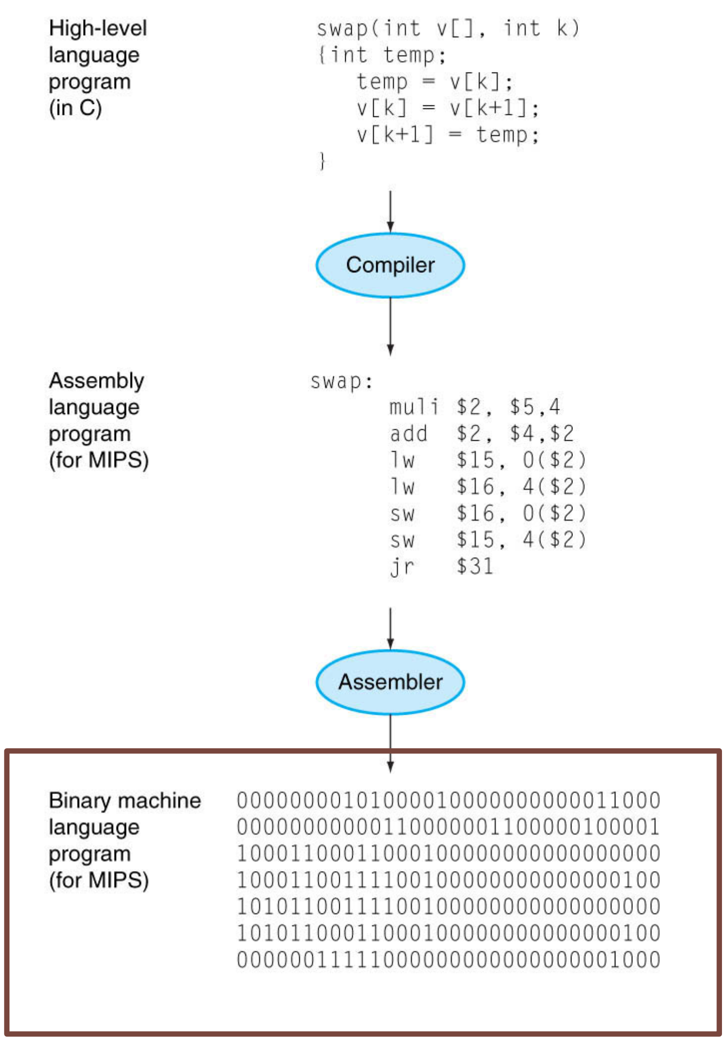
2. Brief Recap



- Compiler translates to assembly language (e.g., **MIPS**)

```
      beq $16, $0, Else
      lw  $8, 4($17)
      add $8, $8, $16
      sw  $8, 0($17)
Else:
```

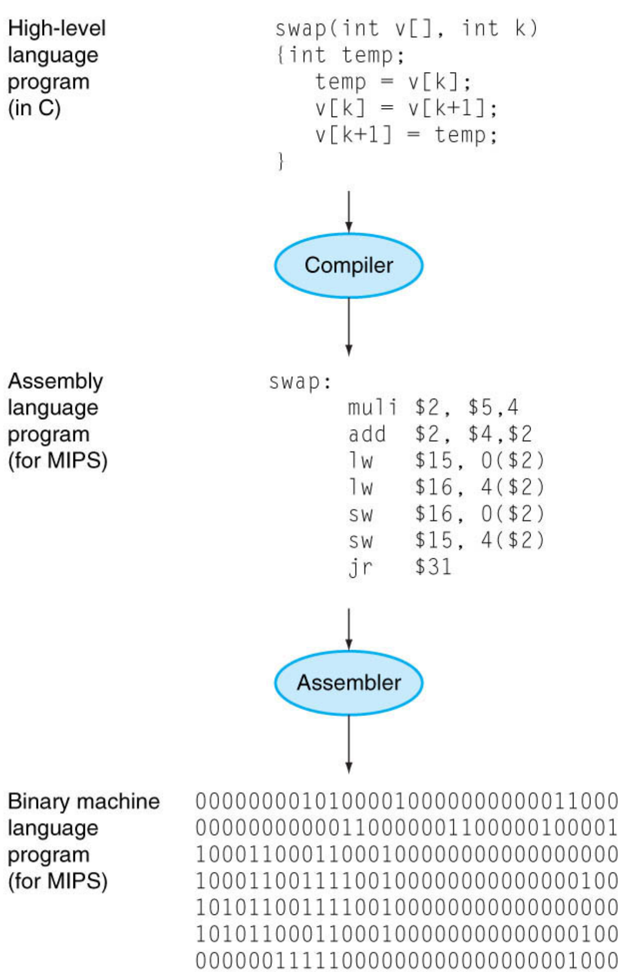
2. Brief Recap



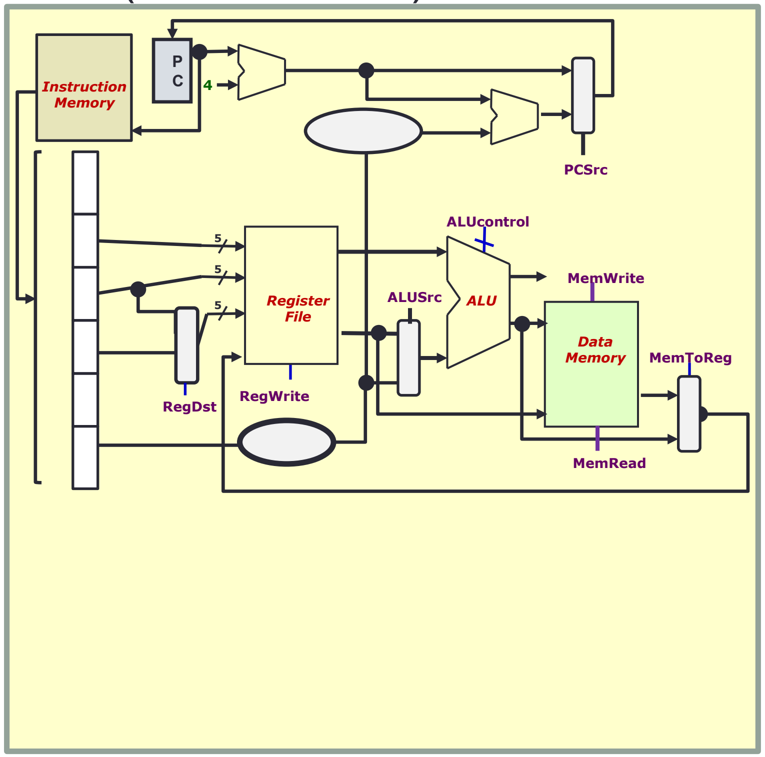
- Assembler translates to machine code (i.e., **binaries**)

```
0001 0010 0000 0000
0000 0000 0000 0011

1000 1110 0010 1000
0000 0000 0000 0100

0000 0010 0000 1000
0100 0000 0001 0100

1010 1110 0010 1000
0000 0000 0000 0000
```

2. Brief Recap



- **Processor** executes the machine code (i.e., **binaries**)



3. From C to Execution

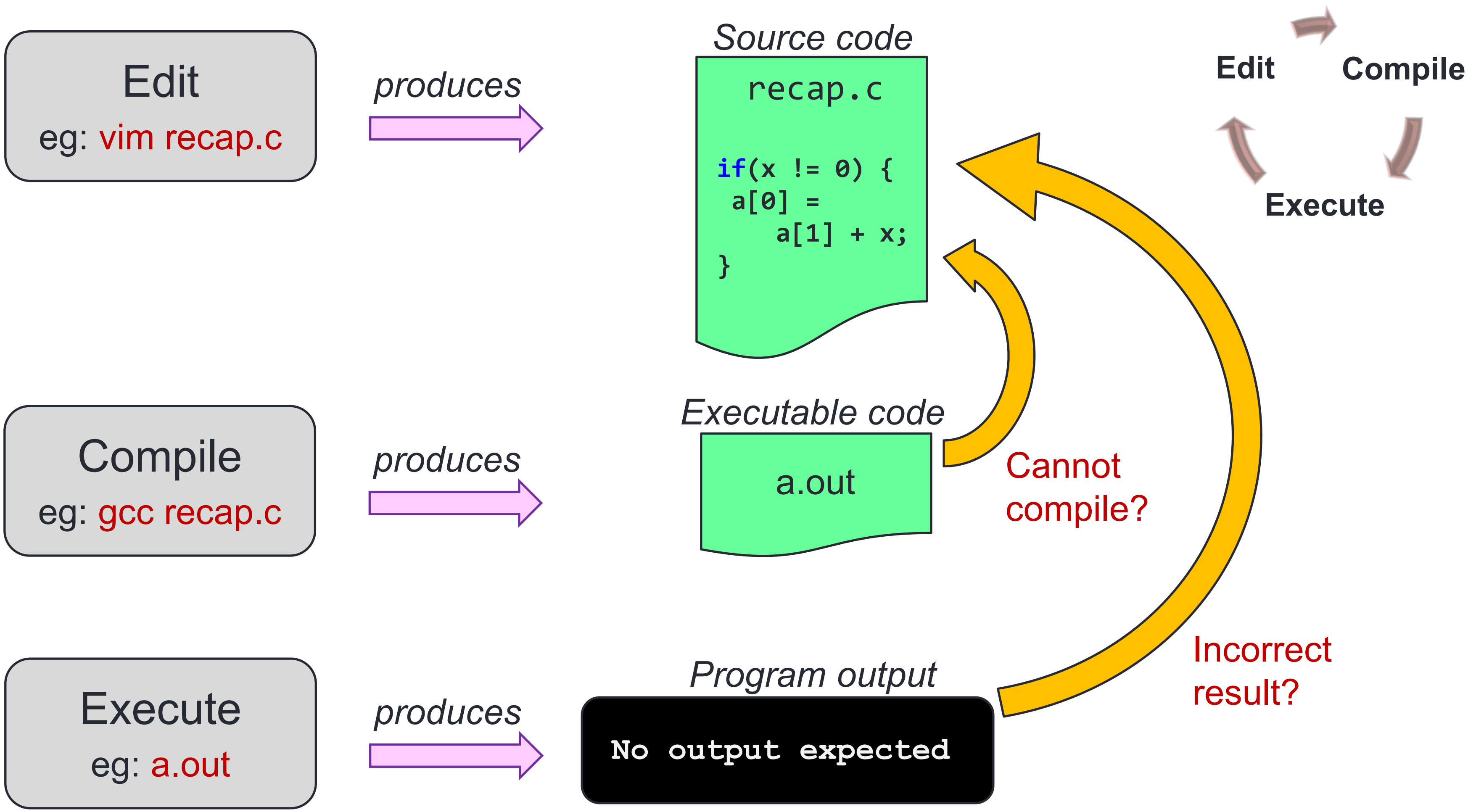
- We play the role of Programmer, Compiler, Assembler, and Processor
  - Program:

```
if(x != 0) {
  a[0] = a[1] + x;
}
```

  - Programmer:
    - Show the workflow of compiling, assembling, and executing C program
  - Compiler:
    - Show how the program is compiled into MIPS
  - Assembler:
    - Show how the MIPS is translated into binaries
  - Processor:
    - Show how the datapath is activated in the processor

3.1 Writing C Program

- Edit, Compile, Execute: Lecture #2, Slide 5



recap.c

```
if(x != 0) {
 a[0] =
    a[1] + x;
}
```

3.2 Compiling to MIPS

- Key Idea #1:
  Compilation is a *structured process*

```
if(x != 0) {
   a[0] = a[1] + x;
}
```

  - Each structure can be compiled *independently*

Inner Structure

```
   a[0] = a[1] + x;
```

Outer Structure

```
if(x != 0) {


}
```

```
recap.c

if(x != 0) {
 a[0] =
    a[1] + x;
}
```

3.2 Compiling to MIPS

- Key Idea #2:
  Variable-to-Register Mapping

```
if(x != 0) {
  a[0] = a[1] + x;
}
```

  - Let the mapping be:

Variable	Register Name	Register Number
x	$s0	$16
a	$s1	$17

Mapping:

```
x: $16
a: $17
```

recap.c

```
if(x != 0) {
 a[0] =
    a[1] + x;
}
```

3.2 Compiling to MIPS

- Common Technique #1:
  Invert the condition for shorter code
  (Lecture #8, Slide 22)

Outer Structure

```
if(x != 0) {


}
```

Outer MIPS Code

```
        beq $16, $0, Else

        # Inner Structure

Else:
```

Mapping:

```
x: $16
a: $17
$t1: $8
```

recap.c

```
if(x != 0) {
 a[0] =
    a[1] + x;
}
```

3.2 Compiling to MIPS

- Common Technique #2:
  Break complex operations, use temp register
  (Lecture #7, Slide 29)

Inner Structure

```
a[0] = a[1] + x;
```

Simplified Inner Structure

```
$t1  = a[1];
$t1  = $t1 + x;
a[0] = $t1;
```

Mapping:

```
x: $16
a: $17
$t1: $8
```

recap.c

```
if(x != 0) {
 a[0] =
    a[1] + x;
}
```

3.2 Compiling to MIPS

- Common Technique #3:
  Array access is `lw`, array update is `sw`
  (Lecture #8, Slide 13)

Simplified Inner Structure

```
$t1  = a[1];
$t1  = $t1 + x;
a[0] = $t1;
```

Inner MIPS Code

```
lw  $8, 4($17)
add $8, $8, $16
sw  $8, 0($17)
```

Mapping:

```
x: $16
a: $17
$t1: $8
```

recap.c

```
if(x != 0) {
 a[0] =
    a[1] + x;
}
```

3.2 Compiling to MIPS

- Common Error #1:
  Assume that the address of the next word can be found by incrementing the address in a register by 1 instead of by the word size in bytes
  - Example:
    - `$t1  = a[1];`
      is translated to
      `lw  $8, 4($17)`
      instead of
      `lw  $8, 1($17)`

Mapping:

```
x: $16
a: $17
$t1: $8
```

recap.c

```c
if(x != 0) {
 a[0] =
    a[1] + x;
}
```

3.2 Compiling to MIPS

- Last Step:
  Combine the two structures logically

Inner MIPS Code

```
lw  $8, 4($17)
add $8, $8, $16
sw  $8, 0($17)
```

Outer MIPS Code

```
      beq $16, $0, Else

      # Inner Structure

Else:
```

Combined MIPS Code

```
      beq $16, $0, Else
      lw  $8, 4($17)
      add $8, $8, $16
      sw  $8, 0($17)
Else:
```

```
recap.mips

      beq $16, $0, Else
      lw  $8, 4($17)
      add $8, $8, $16
      sw  $8, 0($17)
Else:
```

3.3 Assembling to Binary

- Instruction Types Used:
  1. `R-Format:` (Lecture #9, Slide 8)
     - `opcode $rd, $rs, $rt`

6	5	5	5	5	6
opcode	rs	rt	rd	shamt	funct

  2. `I-Format:` (Lecture #9, Slide 14)
     - `opcode $rt, $rs, immediate`

6	5	5	16
opcode	rs	rt	immediate

  3. `Branch:` (Lecture #9, Slide 22)
     - Uses I-Format
     - **PC = (PC + 4) + (`immediate` × 4)**

```
recap.mips

      beq $16, $0, Else
      lw  $8, 4($17)
      add $8, $8, $16
      sw  $8, 0($17)
Else:
```

3.3 Assembling to Binary

- **beq $16, $0, Else**
  - Compute immediate value (Lecture #9, Slide 27)
    - **immediate** = 3
  - Fill in fields (*refer to MIPS Reference Data*)

6	5	5	16
4	16	0	3

  - Convert to binary

000100	10000	00000	0000000000000011

```
      beq $16, $0, Else
      lw  $8, 4($17)
      add $8, $8, $16      +3
      sw  $8, 0($17)
Else:
```

```
recap.mips

      beq $16, $0, Else
      lw  $8, 4($17)
      add $8, $8, $16
      sw  $8, 0($17)
Else:
```

3.3 Assembling to Binary

- `lw   $8, 4($17)`
  - Fill in fields (*refer to MIPS Reference Data*)

6	5	5	16
35	17	8	4

  - Convert to binary

100011	10001	01000	0000000000000100

```
0001 0010 0000 0000 0000 0000 0000 0011
      lw  $8, 4($17)
      add $8, $8, $16
      sw  $8, 0($17)
Else:
```

```
recap.mips

    beq $16, $0, Else
    lw  $8, 4($17)
    add $8, $8, $16
    sw  $8, 0($17)
Else:
```

3.3 Assembling to Binary

- `add $8, $8, $16`
  - Fill in fields (*refer to MIPS Reference Data*)

6	5	5	5	5	6
0	8	16	8	0	32

  - Convert to binary

000000	01000	10000	01000	00000	100000

```
0001 0010 0000 0000 0000 0000 0000 0011
1000 1110 0010 1000 0000 0000 0000 0100
      add $8, $8, $16
      sw  $8, 0($17)
Else:
```

recap.mips

```
    beq $16, $0, Else
    lw  $8, 4($17)
    add $8, $8, $16
    sw  $8, 0($17)
Else:
```

3.3 Assembling to Binary

- **sw $8, 0($17)**
  - Fill in fields (*refer to MIPS Reference Data*)

6	5	5	16
43	17	8	0

  - Convert to binary

101011	10001	01000	0000000000000000

```
0001 0010 0000 0000 0000 0000 0000 0011
1000 1110 0010 1000 0000 0000 0000 0100
0000 0001 0001 0000 0100 0000 0010 0000
       sw  $8, 0($17)
Else:
```

recap.mips

```
      beq $16, $0, Else
      lw  $8, 4($17)
      add $8, $8, $16
      sw  $8, 0($17)
Else:
```

3.3 Assembling to Binary

- Final Binary
  - Hard to read?
  - Don’t worry, this is intended for machine not for human!

```
0001 0010 0000 0000 0000 0000 0000 0011
1000 1110 0010 1000 0000 0000 0000 0100
0000 0001 0001 0000 0100 0000 0010 0000
1010 1110 0010 1000 0000 0000 0000 0000
```

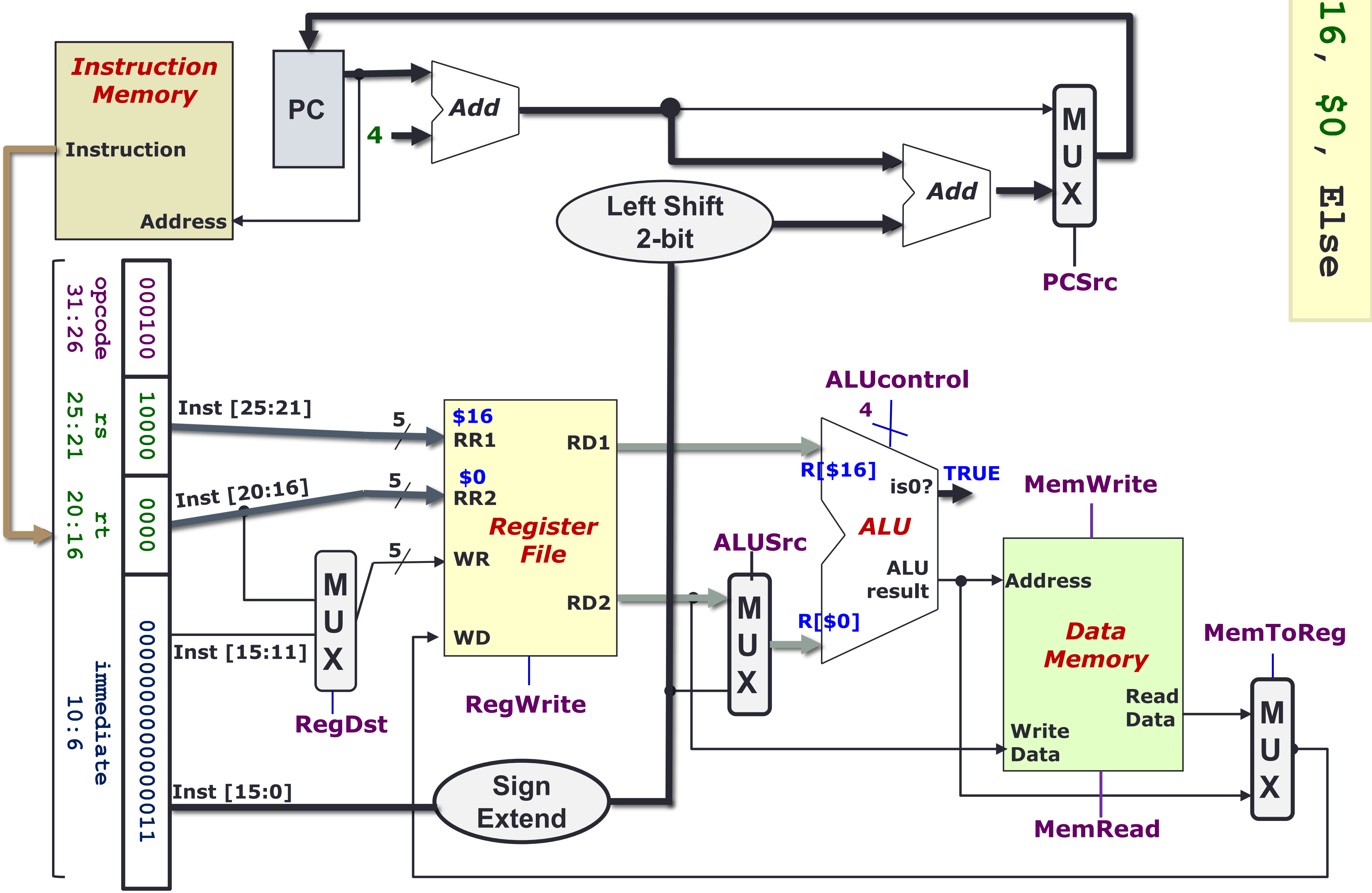
3.4 Execution (Datapath)

- Given the binary
  - Assume two possible executions:
    1. `$16 == $0` *(shorter)*
    2. `$16 != $0` *(longer)*
  - Convention:

Fetch:	→	Memory:	→
Decode:	→	Reg Write:	→
ALU:	→	Other:	→

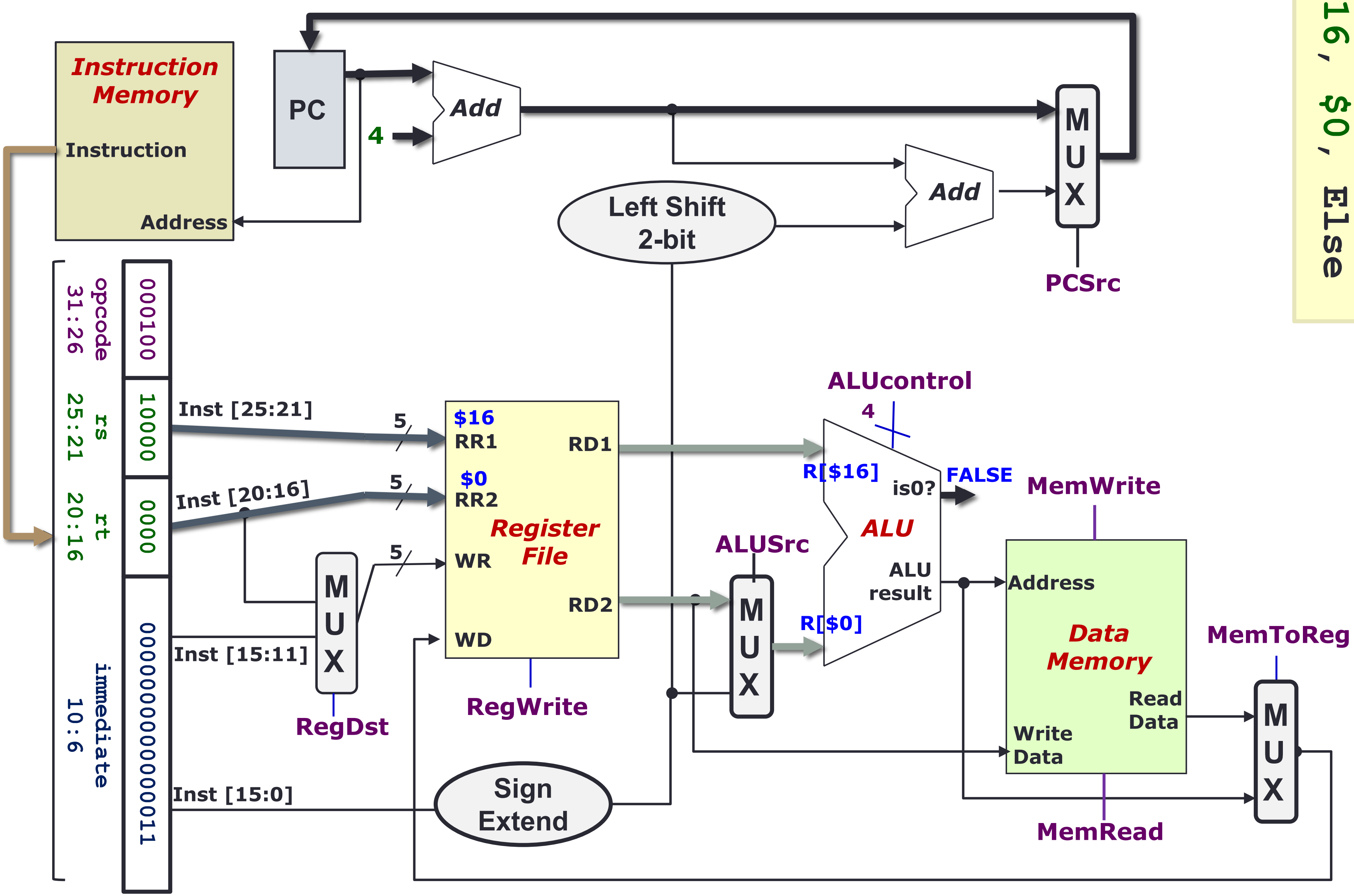
```
0001 0010 0000 0000 0000 0000 0000 0011
1000 1110 0010 1000 0000 0000 0000 0100
0000 0001 0001 0000 0100 0000 0010 0000
1010 1110 0010 1000 0000 0000 0000 0000
```

- Assume $16 == $0


Instruction Memory
Instruction
Address
PC
4
Add
Add
Left Shift 2-bit
M U X
PCSrc
opcode 31:26
000100
rs 25:21
10000
rt 20:16
0000
immediate 10:6
0000000000000011
Inst [25:21]
Inst [20:16]
Inst [15:11]
Inst [15:0]
5
5
5
M U X
RegDst
$16
RR1
RD1
$0
RR2
Register File
WR
RD2
WD
RegWrite
Sign Extend
ALUcontrol
4
R[$16]
is0?
TRUE
ALU
ALU result
R[$0]
ALUSrc
M U X
MemWrite
Address
Data Memory
Read Data
Write Data
MemRead
MemToReg
M U X
beq $16, $0, Else

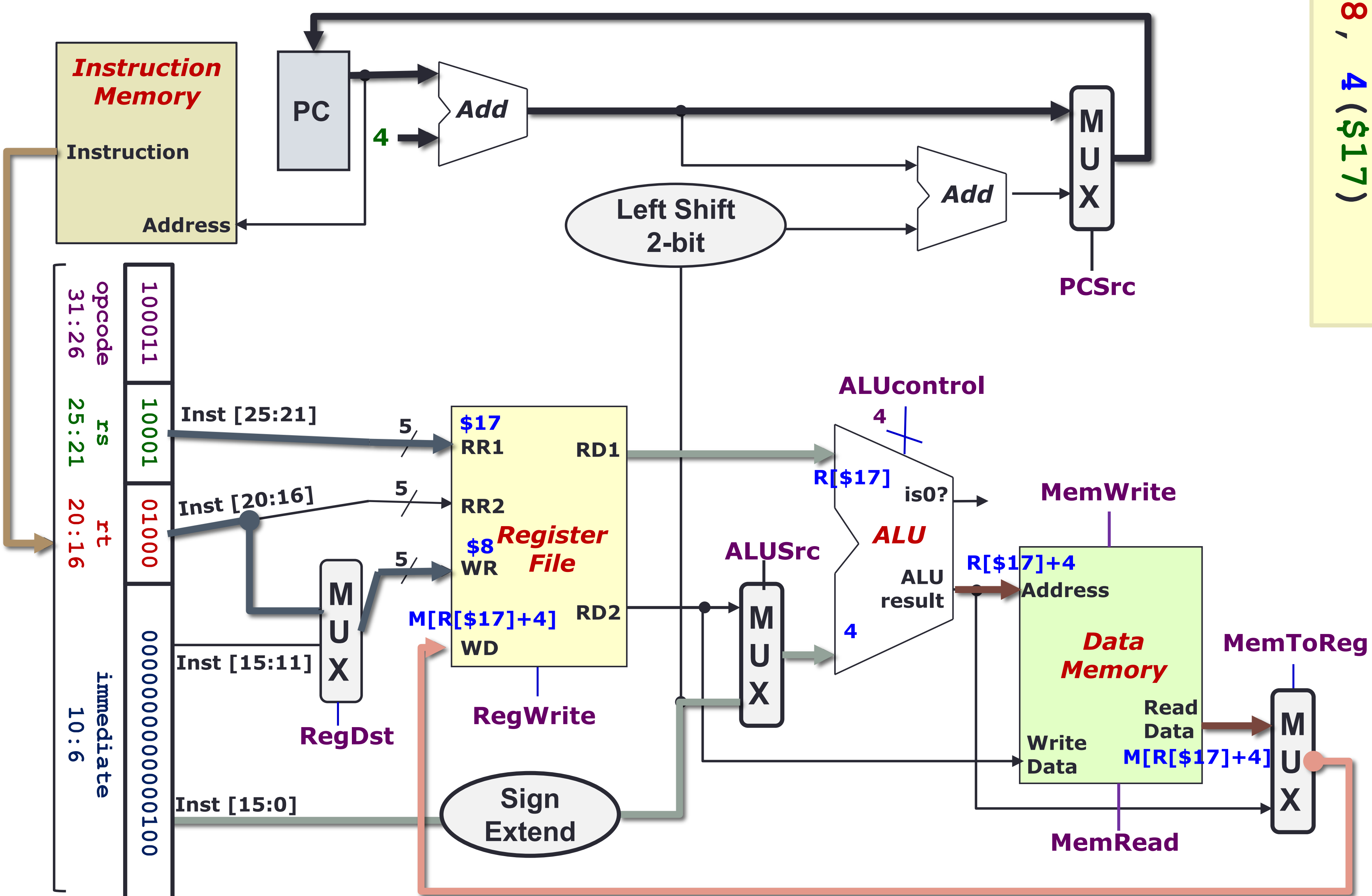
- Assume `$16 != $0`

`beq $16, $0, Else`

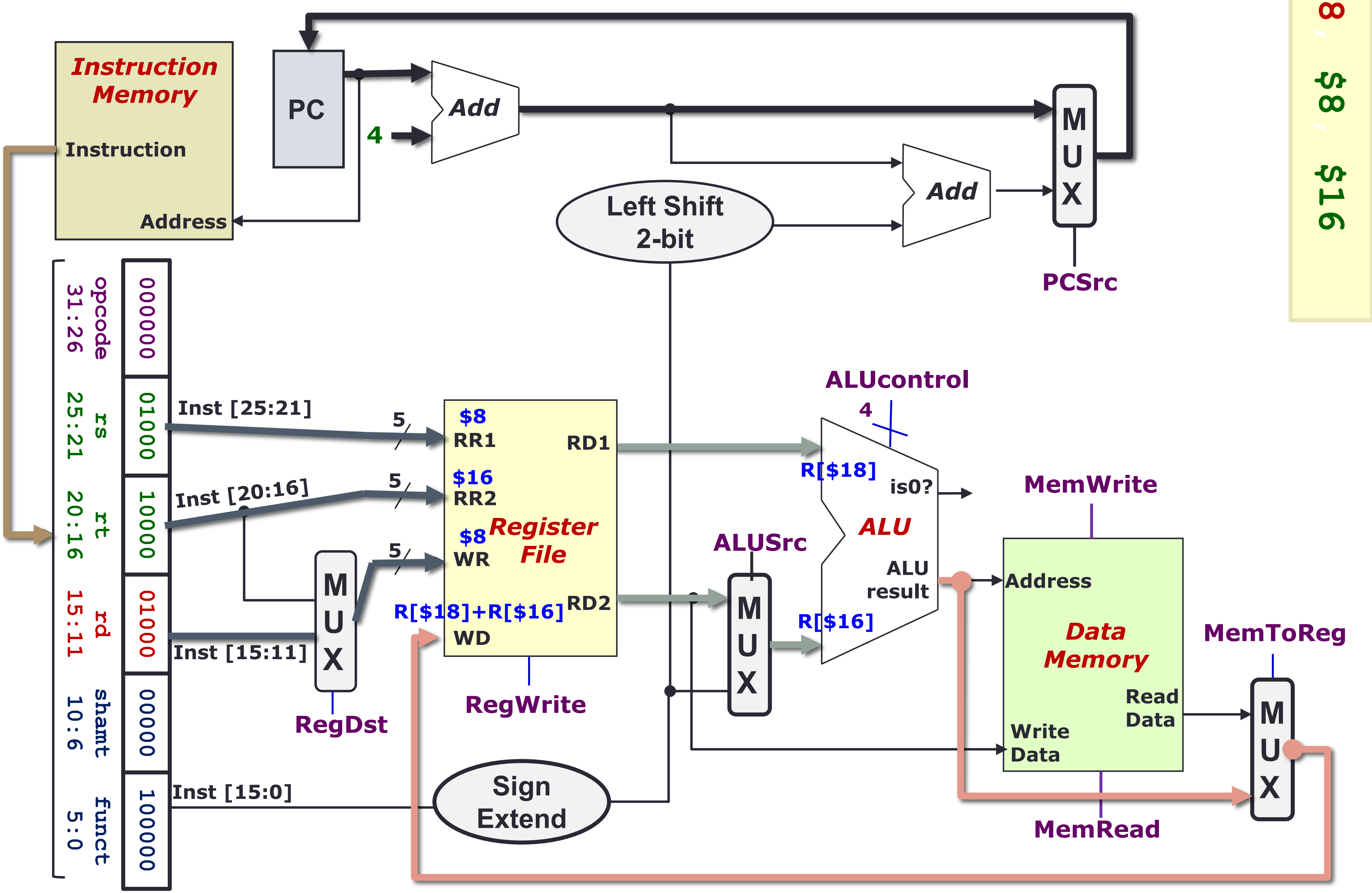


- Assume $16 != $0

lw $8, 4($17)

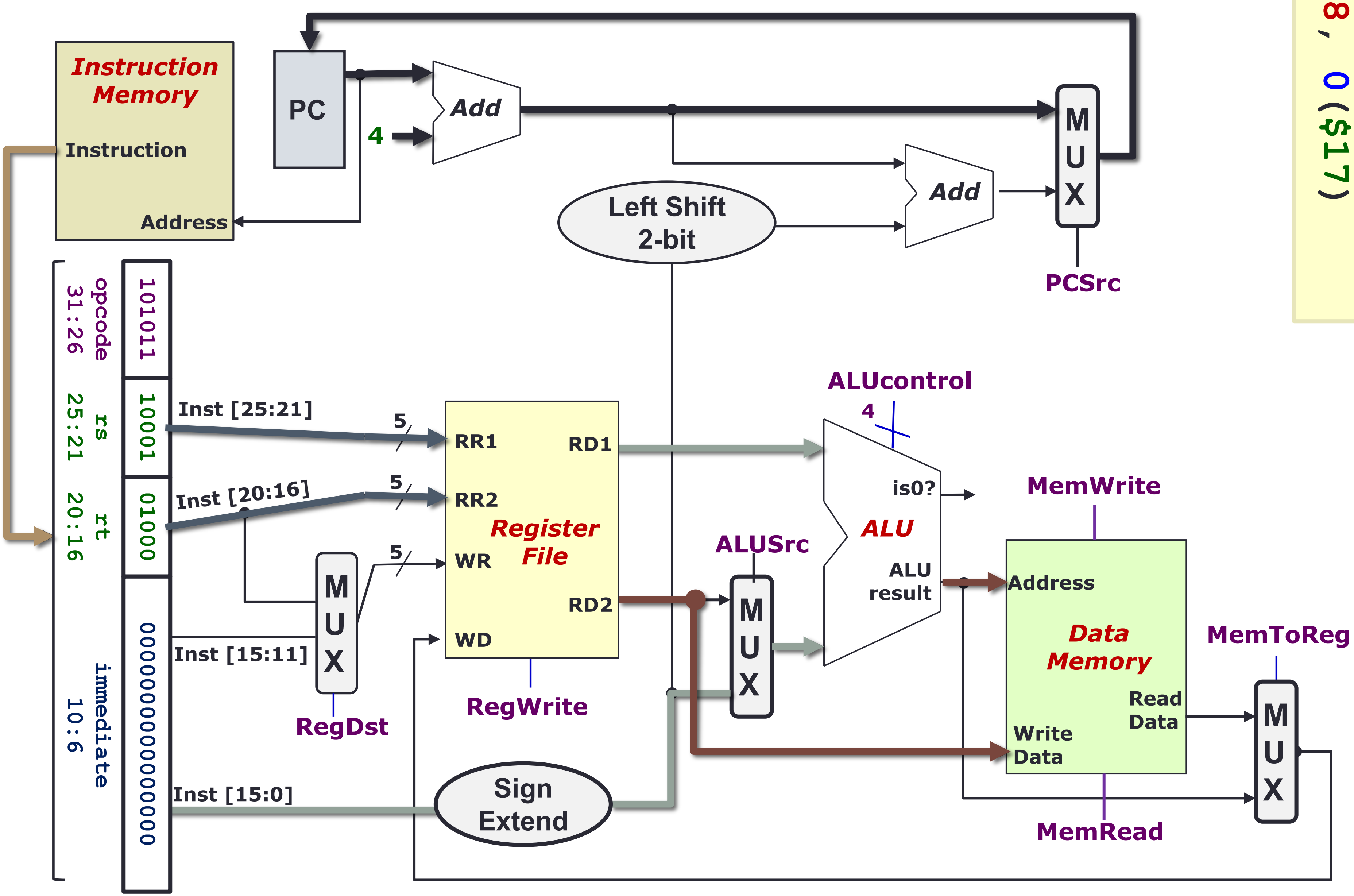


- Assume `$16 != $0`


add $8, $8, $16
Instruction Memory
Instruction
Address
PC
4
Add
Add
Left Shift 2-bit
MUX
PCSrc
opcode 31:26 000000
rs 25:21 01000
rt 20:16 10000
rd 15:11 01000
shamt 10:6 00000
funct 5:0 100000
Inst [25:21]
Inst [20:16]
Inst [15:11]
Inst [15:0]
5
5
5
MUX
RegDst
$8
RR1
$16
RR2
$8
WR
R[$18]+R[$16]
WD
RD1
RD2
Register File
RegWrite
Sign Extend
ALUcontrol
4
R[$18]
is0?
ALU
ALU result
R[$16]
ALUSrc
MUX
MemWrite
Address
Data Memory
Read Data
Write Data
MemRead
MemToReg
MUX

- Assume `$16 != $0`

`sw $8, 0($17)`



End of File