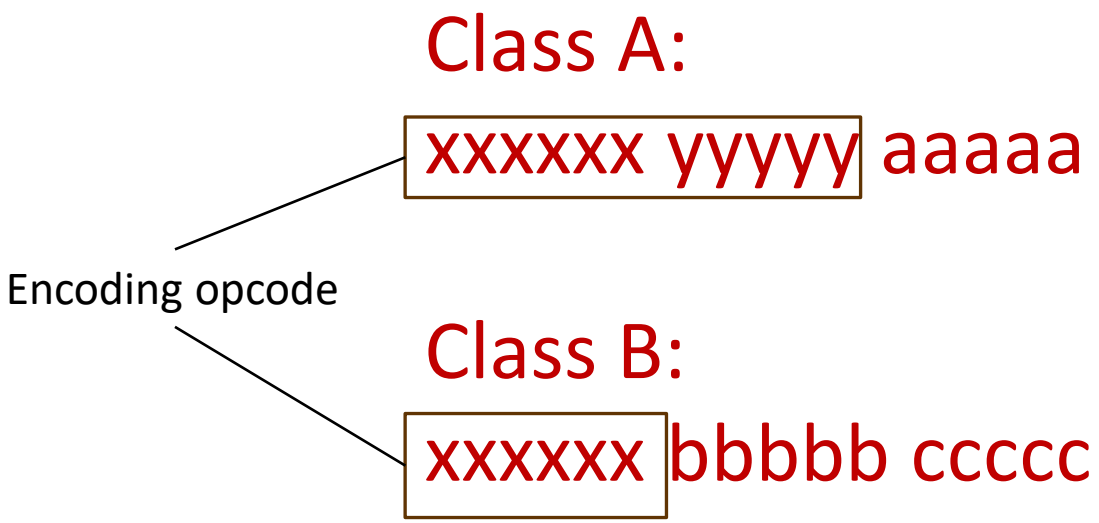
# CS2100

TUTORIAL #4

## DATA AND CONTROL PATH

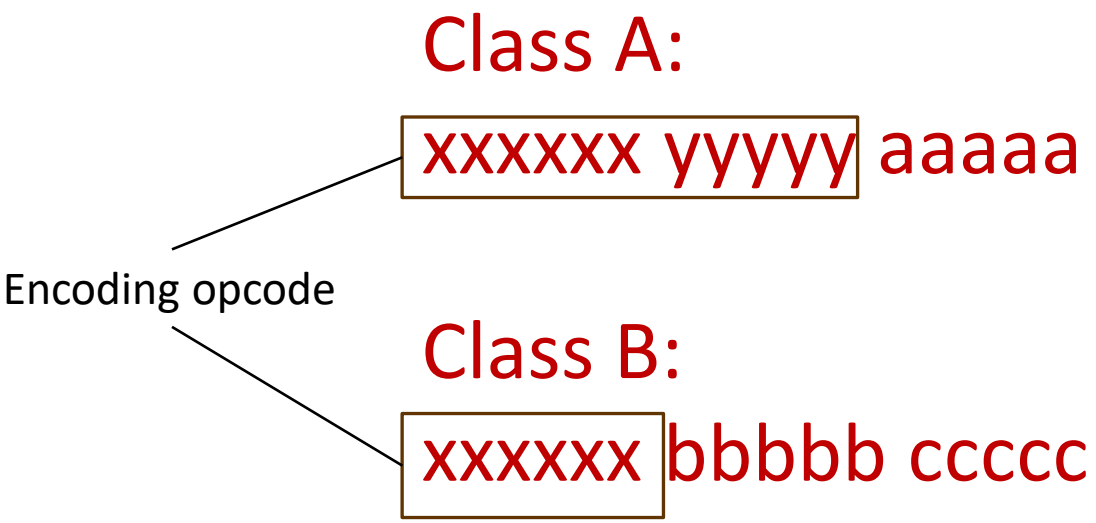
Q1. An ISA has 16-bit instructions and 5-bit addresses. There are two classes of instructions:
Class A instructions have one address,
Class B instructions have two addresses.
Both classes exist and the encoding space for opcode is fully utilized.
(a) What is the minimum total number of instructions?




64 unique values can be represented by xxxxxx.
Minimize total number of instructions, assign 1 unique value of xxxxxx to class A and 63 unique values of xxxxxx to class B.

Number of instructions in class B = 63
Number of instructions in class A = 1*32 = 32
Minimum total no. of instr = 63 + 32 = 95

Q1. An ISA has 16-bit instructions and 5-bit addresses. There are two classes of instructions:
Class A instructions have one address,
Class B instructions have two addresses.
Both classes exist and the encoding space for opcode is fully utilized.
(b) What is the maximum total number of instructions?




64 unique values can be represented by xxxxxx.
Maximize total number of instructions, assign 1 unique value of xxxxxx to class B and 63 unique values of xxxxxx to class A.

No. of instructions in class B = 1
N0. of instructions in class A = 63*32 = 2016
Maximum total no. of instr = 2016 + 1 = 2017

Q2. Implement the pseudo-instruction as real MIPS instructions

(a) `bgt $r1, $r2, L`

This is equivalent to

```
blt $r2, $r1, L
```

Which can be implemented with

```
slt $at, $r2, $r1
bne $at, $zero, L
```

Temporary register reserved for the assembler

**REGISTER NAME, NUMBER, USE, CALL CONVENTION**

| NAME | NUMBER | USE | PRESERVED ACROSS A CALL? |
|---|---|---|---|
| $zero | 0 | The Constant Value 0 | N.A. |
| $at | 1 | Assembler Temporary | No |
| $v0-$v1 | 2-3 | Values for Function Results and Expression Evaluation | No |
| $a0-$a3 | 4-7 | Arguments | No |
| $t0-$t7 | 8-15 | Temporaries | No |
| $s0-$s7 | 16-23 | Saved Temporaries | Yes |
| $t8-$t9 | 24-25 | Temporaries | No |
| $k0-$k1 | 26-27 | Reserved for OS Kernel | No |
| $gp | 28 | Global Pointer | Yes |
| $sp | 29 | Stack Pointer | Yes |
| $fp | 30 | Frame Pointer | Yes |
| $ra | 31 | Return Address | Yes |

Q2. Implement the pseudo-instruction as real MIPS instructions

(b) `bge $r1, $r2, L`

```
if (r1 >= r2) goto L
```

This is equivalent to

```
if (!(r1 < r2)) goto L
```

```
slt $at, $r1, $r2
beq $at, $zero, L
```

Q2. Implement the pseudo-instruction as real MIPS instructions

(c) `ble $r1, $r2, L`

```
if (r1 <= r2) goto L
```

This is equivalent to

```
if (!(r2 < r1)) goto L
```

```
slt $at, $r2, $r1
beq $at, $zero, L
```

Q2. Implement the pseudo-instruction as real MIPS instructions

(d) `li $r, imm`

Case 1: imm can be represented using 16-bits

```
ori $r, $zero, imm
```

Case 2: imm cannot be represented using 16-bits

```
lui $r, imm_Upper
ori $r, $r, imm_Lower
```

Can we use addi instead of ori?

No. addi uses sign extend imm while ori uses zero extend imm. If bit 15 of $imm_{Lower}$ is 1, then ori and addi gives different results.

Q2. Implement the pseudo-instruction as real MIPS instructions

(e) nop

Possible answers:

```
add $r, $r, $zero
sub $r, $r, $zero
ori $r, $r, 0
sll $zero, $zero, 0
```

Prof. Wong likes this because the encoding gives 0x00000000

# Q3. lw $24, 0($15)

(a) 10 0011 01111 11000 0000 0000 0000 0000 = 0x8DF80000

(b)

| RegDst | RegWr | ALUSrc | MRd | MWr | MtoR | Brch | ALUop | ALUctrl |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 00 | 0010 |

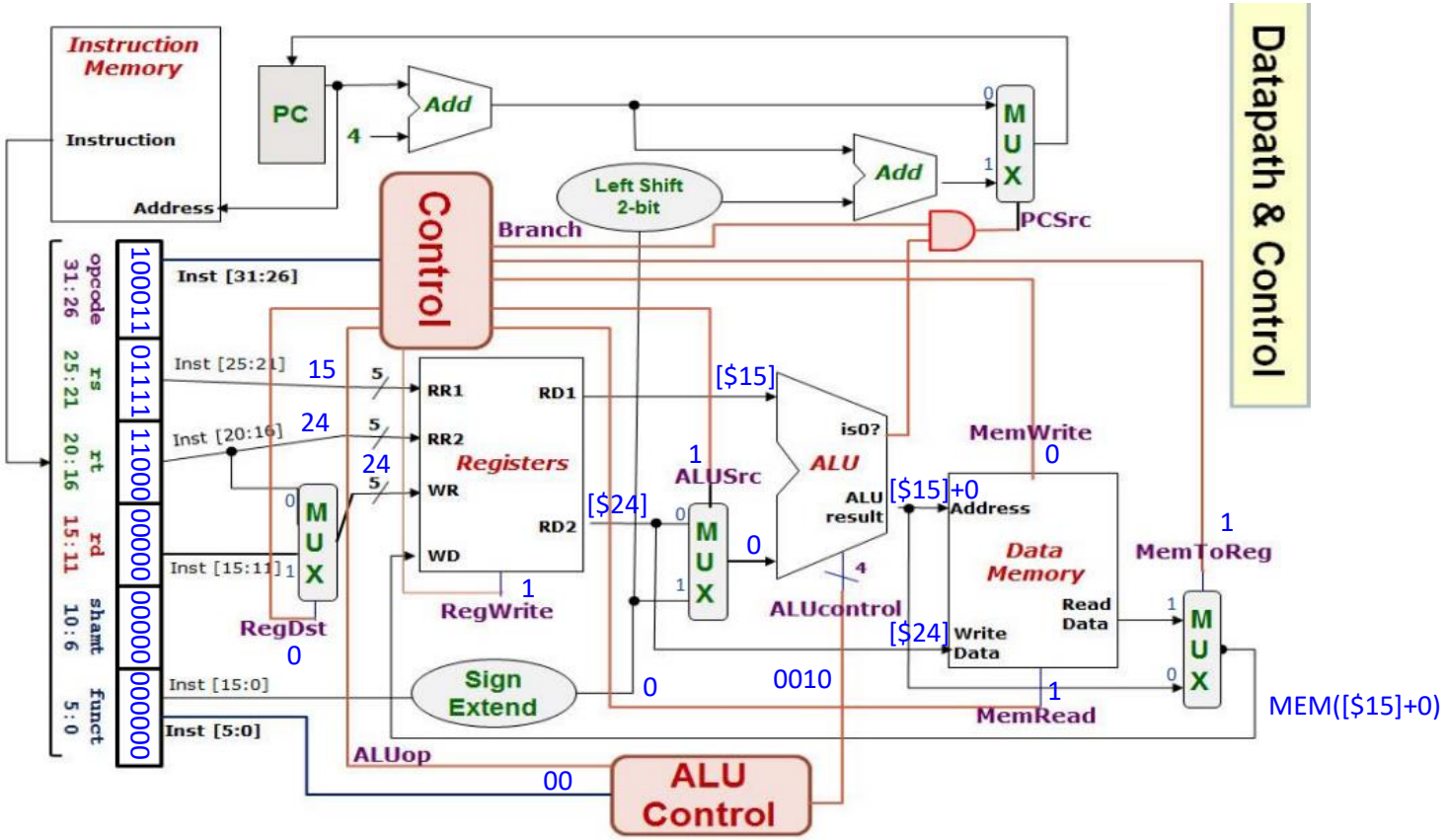
| | RegDst | ALUSrc | MemTo Reg | Reg Write | Mem Read | Mem Write | Branch | ALUop | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | op1 | op0 |
| R-type | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| lw | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| sw | X | 1 | X | 0 | 0 | 1 | 0 | 0 | 0 |
| beq | X | 0 | X | 0 | 0 | 0 | 1 | 0 | 1 |

| Opcode | ALUop | Instruction Operation | Funct field | ALU action | ALU control |
|---|---|---|---|---|---|
| lw | 00 | load word | xxxxxx | add | 0010 |
| sw | 00 | store word | xxxxxx | add | 0010 |
| beq | 01 | branch equal | xxxxxx | subtract | 0110 |
| R-type | 10 | add | 10 0000 | add | 0010 |
| R-type | 10 | subtract | 10 0010 | subtract | 0110 |
| R-type | 10 | AND | 10 0100 | AND | 0000 |
| R-type | 10 | OR | 10 0101 | OR | 0001 |
| R-type | 10 | set on less than | 10 1010 | set on less than | 0111 |

# Q3. lw $24, 0($15)

(b)

| RegDst | RegWr | ALUSrc | MRd | MWr | MtoR | Brch | ALUop | ALUctrl |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 00 | 0010 |



# Q3. `lw $24, 0($15)`

(b)

| Registers File | | | | ALU | | Data Memory | |
|---|---|---|---|---|---|---|---|
| RR1 | RR2 | WR | WD | Opr1 | Opr2 | Address | Write Data |
| $15 | $24 | $24 | MEM([$15]) | [$15] | 0 | [$15] + 0 | [$24] |

(c) Next PC = PC + 4

# Q3. `beq $1, $3, 12`

(a) 00 0100 00001 00011 0000 0000 0000 1100 = 0x1023000C

(b)

| RegDst | RegWr | ALUSrc | MRd | MWr | MtoR | Brch | ALUop | ALUctrl |
|---|---|---|---|---|---|---|---|---|
| X | 0 | 0 | 0 | 0 | X | 1 | 01 | 0110 |

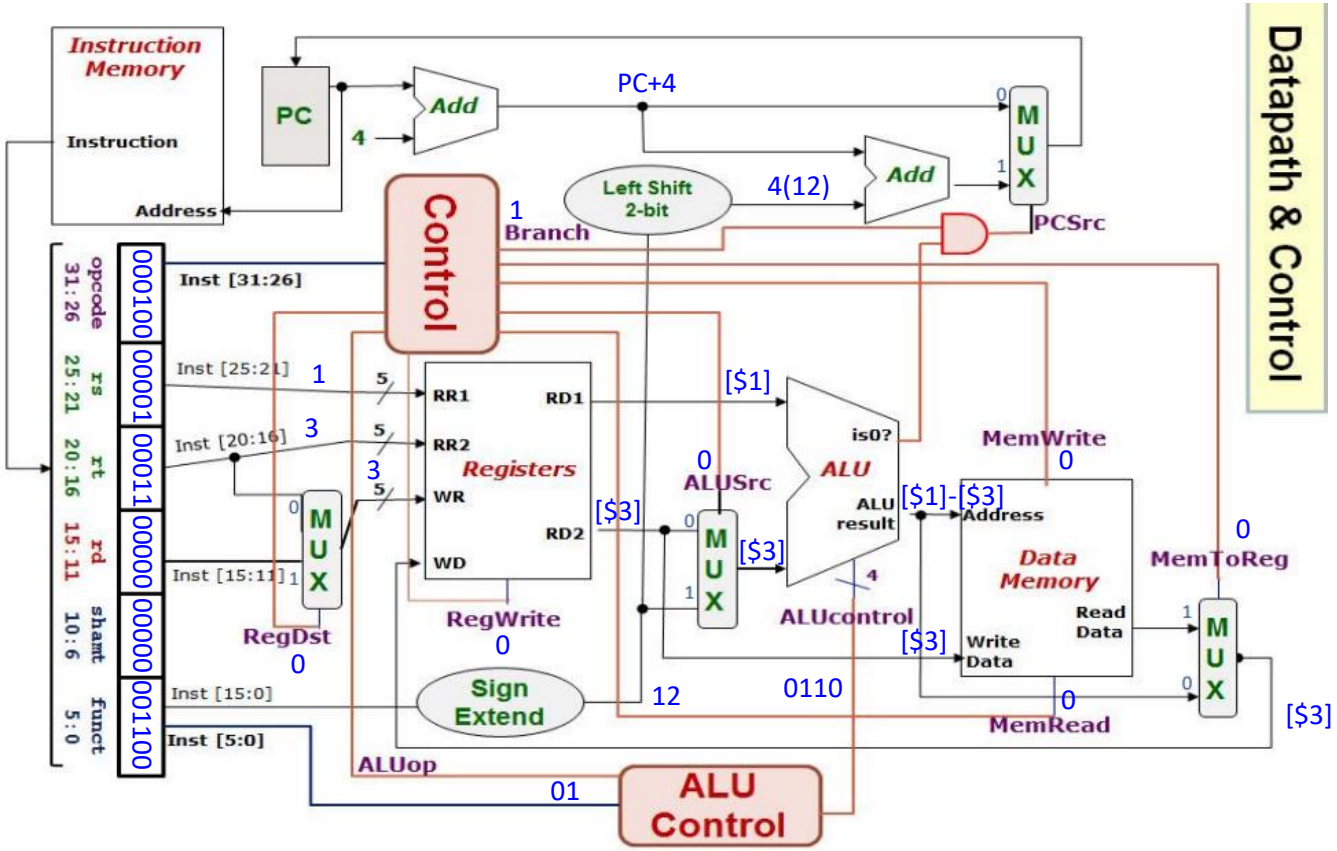
| | RegDst | ALUSrc | MemTo Reg | Reg Write | Mem Read | Mem Write | Branch | ALUop | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | op1 | op0 |
| R-type | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| lw | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| sw | X | 1 | X | 0 | 0 | 1 | 0 | 0 | 0 |
| beq | X | 0 | X | 0 | 0 | 0 | 1 | 0 | 1 |

| Opcode | ALUop | Instruction Operation | Funct field | ALU action | ALU control |
|---|---|---|---|---|---|
| lw | 00 | load word | xxxxxx | add | 0010 |
| sw | 00 | store word | xxxxxx | add | 0010 |
| beq | 01 | branch equal | xxxxxx | subtract | 0110 |
| R-type | 10 | add | 10 0000 | add | 0010 |
| R-type | 10 | subtract | 10 0010 | subtract | 0110 |
| R-type | 10 | AND | 10 0100 | AND | 0000 |
| R-type | 10 | OR | 10 0101 | OR | 0001 |
| R-type | 10 | set on less than | 10 1010 | set on less than | 0111 |

# Q3. beq $1, $3, 12

(b)

| RegDst | RegWr | ALUSrc | MRd | MWr | MtoR | Brch | ALUop | ALUctrl |
|---|---|---|---|---|---|---|---|---|
| X | 0 | 0 | 0 | 0 | X | 1 | 01 | 0110 |



# Q3. `beq $1, $3, 12`

(b)

| Registers File | | | | ALU | | Data Memory | |
|---|---|---|---|---|---|---|---|
| RR1 | RR2 | WR | WD | Opr1 | Opr2 | Address | Write Data |
| $1 | $3 | $3 OR $0 | [$1] – [$3] OR Random value | [$1] | [$3] | [$1] – [$3] | [$3] |

(c) Next PC = PC + 4 OR PC + 4 + 4(12)

# Q3. sub $25, $20, $5

(a) 00 0000 10100 00101 11001 00000 10 0010 = 0x0285C822

(b)

| RegDst | RegWr | ALUSrc | MRd | MWr | MtoR | Brch | ALUop | ALUctrl |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 10 | 0110 |

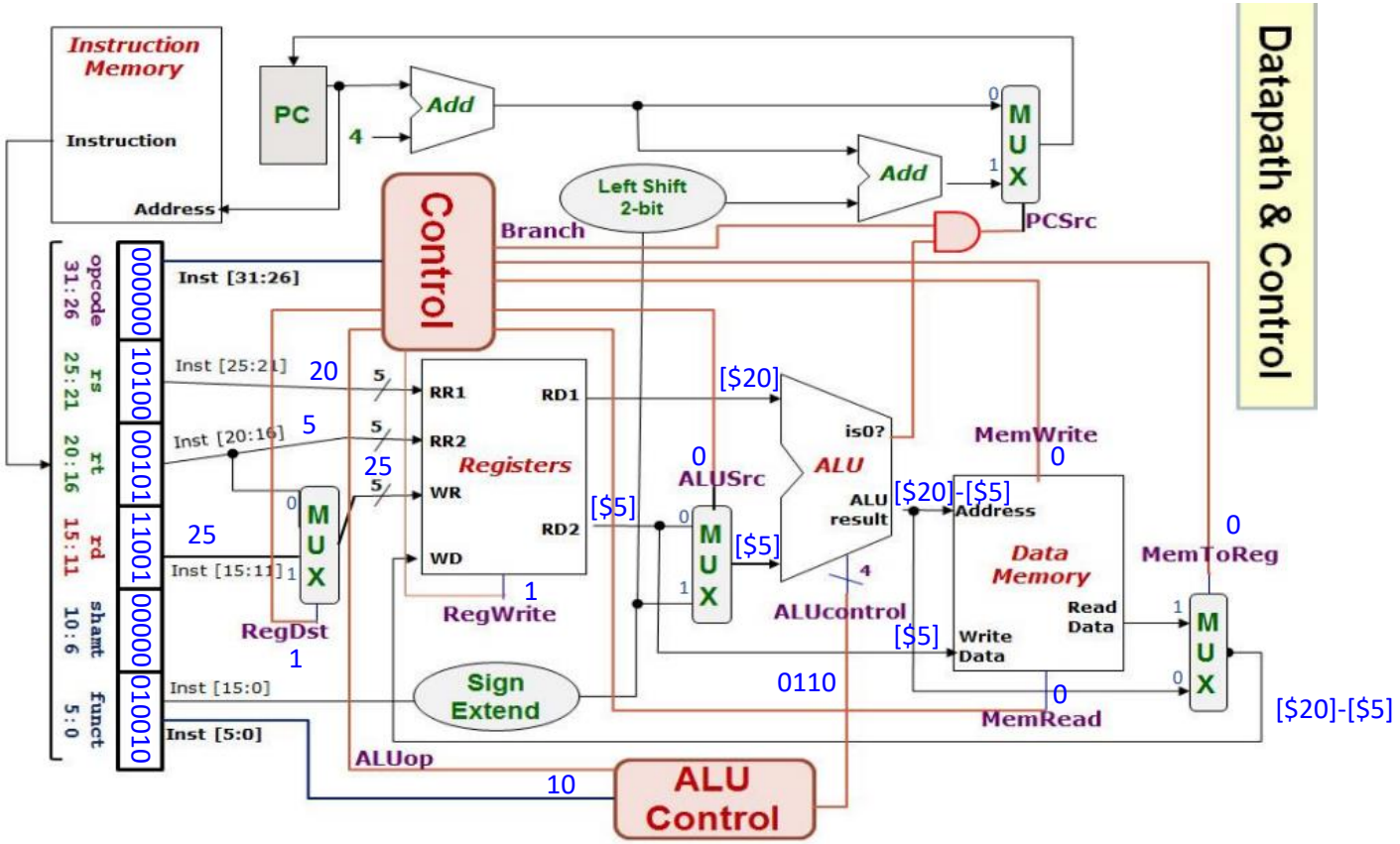
| | RegDst | ALUSrc | MemTo Reg | Reg Write | Mem Read | Mem Write | Branch | ALUop | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | op1 | op0 |
| R-type | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| lw | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| sw | X | 1 | X | 0 | 0 | 1 | 0 | 0 | 0 |
| beq | X | 0 | X | 0 | 0 | 0 | 1 | 0 | 1 |

| Opcode | ALUop | Instruction Operation | Funct field | ALU action | ALU control |
|---|---|---|---|---|---|
| lw | 00 | load word | xxxxxx | add | 0010 |
| sw | 00 | store word | xxxxxx | add | 0010 |
| beq | 01 | branch equal | xxxxxx | subtract | 0110 |
| R-type | 10 | add | 10 0000 | add | 0010 |
| R-type | 10 | subtract | 10 0010 | subtract | 0110 |
| R-type | 10 | AND | 10 0100 | AND | 0000 |
| R-type | 10 | OR | 10 0101 | OR | 0001 |
| R-type | 10 | set on less than | 10 1010 | set on less than | 0111 |

# Q3. sub $25, $20, $5

(b)

| RegDst | RegWr | ALUSrc | MRd | MWr | MtoR | Brch | ALUop | ALUctrl |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 10 | 0110 |



Q3. `sub $25, $20, $5`

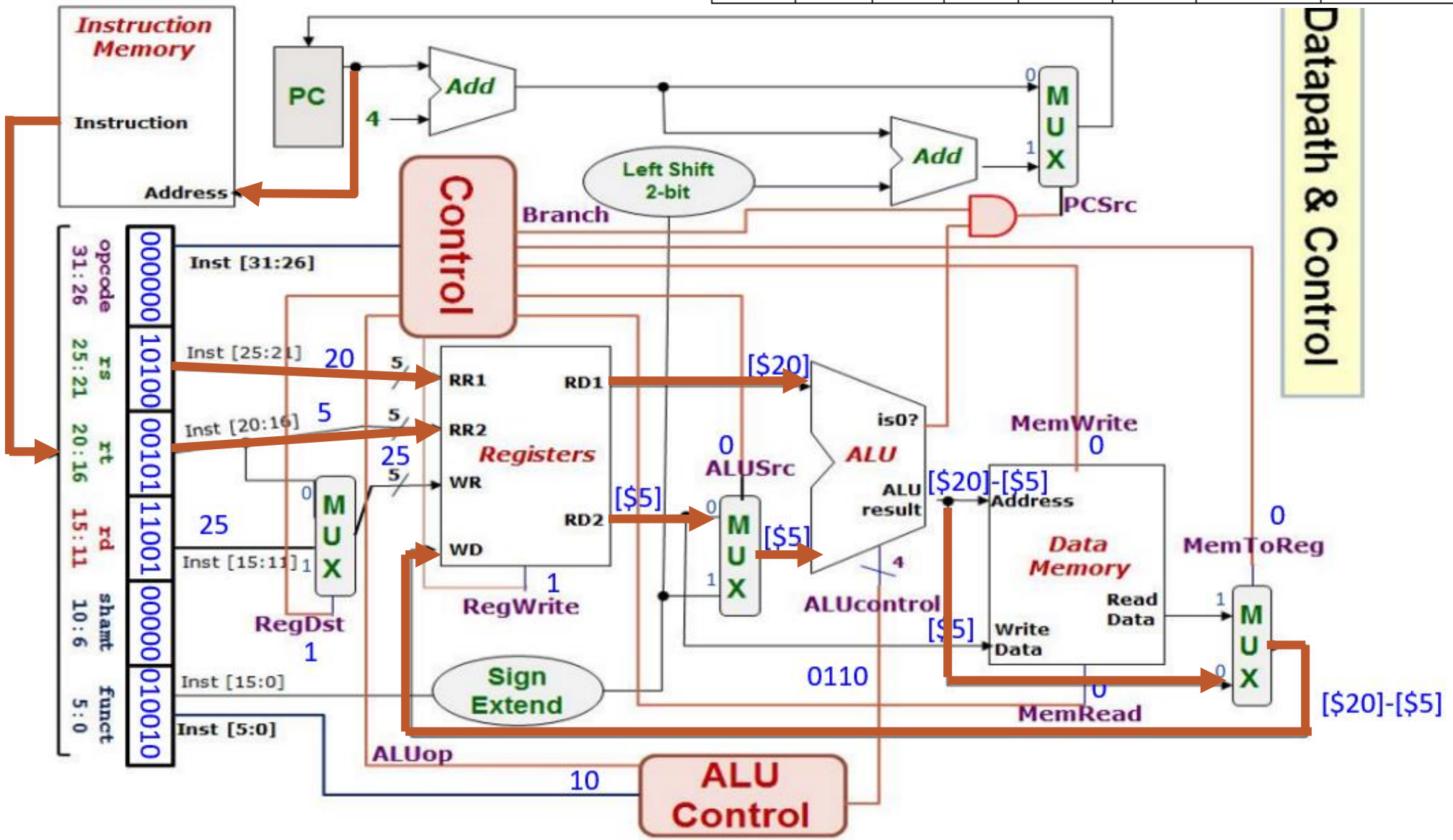
(b)

| Registers File | | | | ALU | | Data Memory | |
|---|---|---|---|---|---|---|---|
| RR1 | RR2 | WR | WD | Opr1 | Opr2 | Address | Write Data |
| $20 | $5 | $25 | [$20] – [$5] | [$20] | [$5] | [$20] – [$5] | [$5] |

(c) Next PC = PC + 4

# Q4a. sub $25, $20, $5

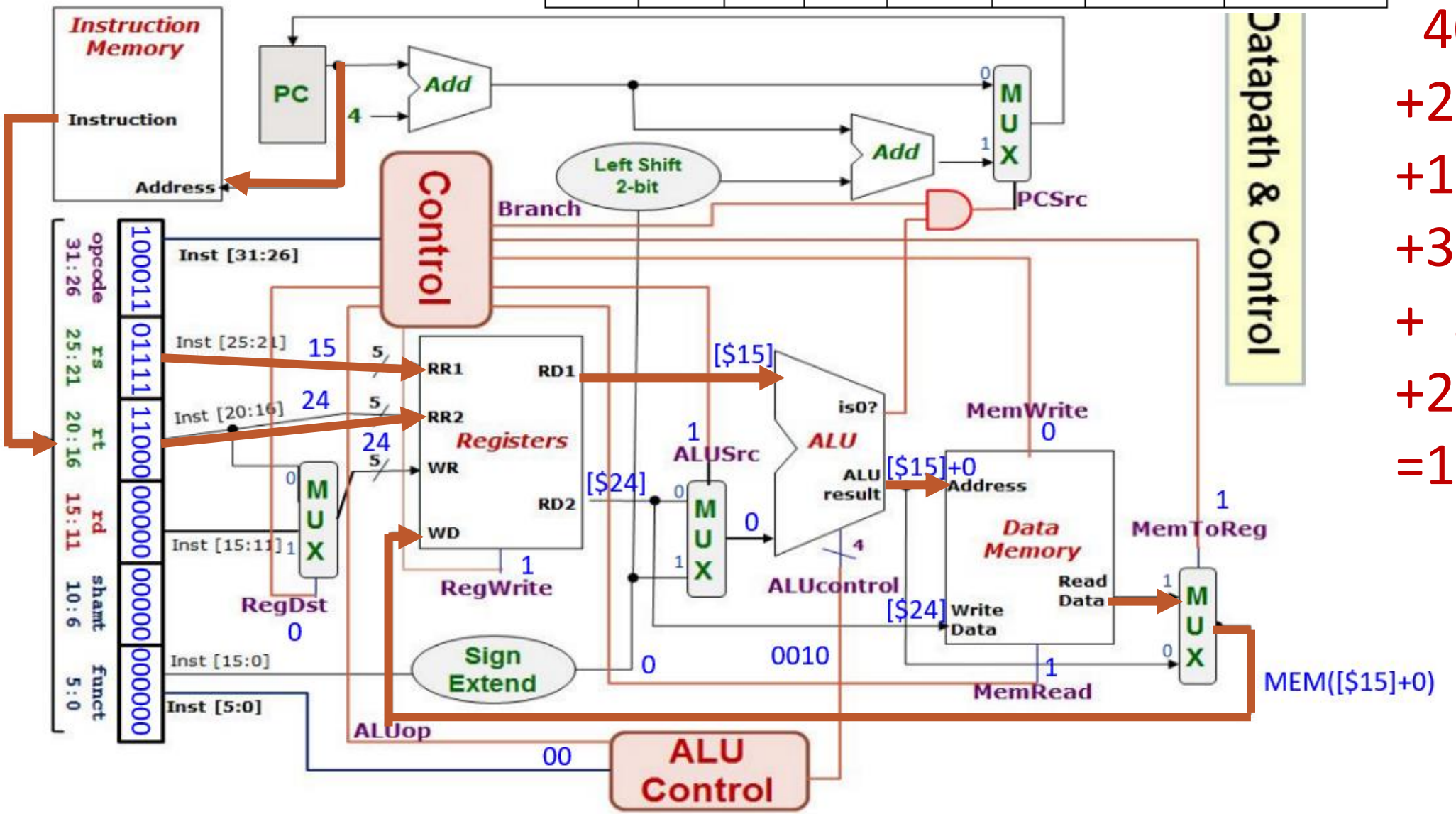
| Inst-Mem | Adder | MUX | ALU | Reg-File | Data-Mem | Control/ ALUControl | Left-shift/ Sign-Extend/ AND |
|---|---|---|---|---|---|---|---|
| 400ps | 100ps | 30ps | 120ps | 200ps | 350ps | 100ps | 20ps |




400
+200
+ 30
+120
+ 30
+200
=980ps

Q4b. `lw $24, 0($15)`

| Inst-Mem | Adder | MUX | ALU | Reg-File | Data-Mem | Control/ ALUControl | Left-shift/ Sign-Extend/ AND |
|---|---|---|---|---|---|---|---|
| 400ps | 100ps | 30ps | 120ps | 200ps | 350ps | 100ps | 20ps |




400
+200
+120
+350
+ 30
+200
=1300ps

END OF FILE

# Additional Question 1

Consider the case of a processor with an instruction length of 16 bits and with 32 general-purpose registers and 256 addresses.

Is it possible to have 30 instructions that operate on one address and one register?

# Additional Answer

Consider the case of a processor with an instruction length of 16 bits and with 32 general-purpose registers and 256 addresses.

Is it possible to have 30 instructions that operate on one address and one register?

No, 8 bits are needed for encoding a single address and 5 bits are needed for encoding a register. That leaves 3 bits for encoding the opcode.

# Additional Question 2

Using the information from Tutorial Q4, what is the critical path of the instruction:

addi $1, $8, 2100

(Note that the datapath from lecture does not actually support I-format instructions. However, students are expected to be able to extend the datapath to support other instructions such as I-format instructions and slt).

Hint: This question does not actually require updating the control signals and datapath. Think about whether there are any instructions that are similar to addi and are already supported by the current datapath.

# Additional Answer 2

Using the information from Tutorial Q4, what is the latency of the instruction:

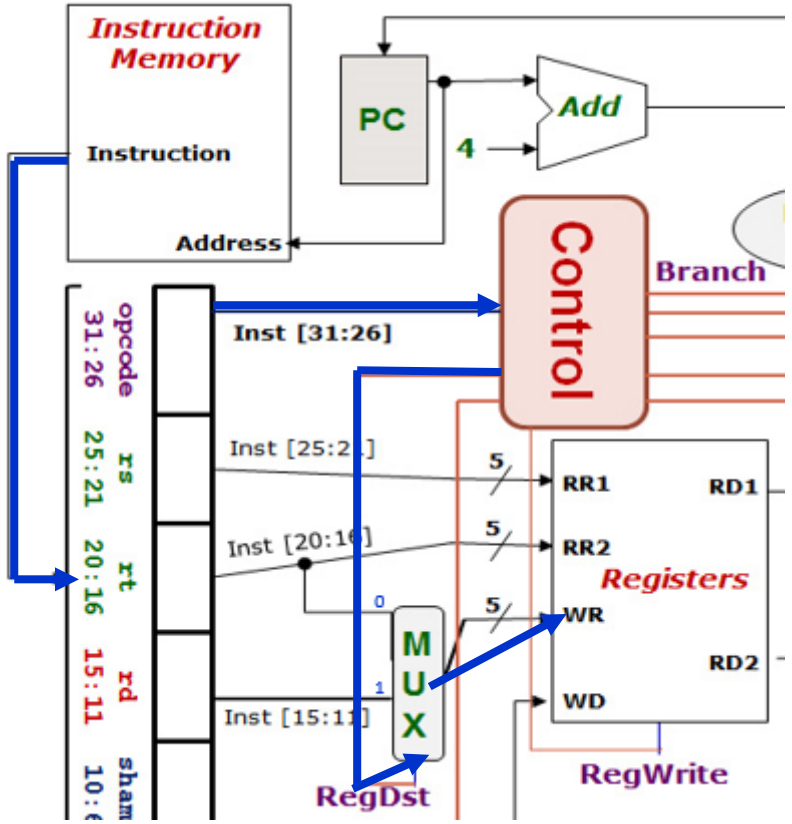
addi $1, $8, 2100

Critical Path:
Instr. Mem (400) -> RegFile (200) -> ALU (120) -> MUX (30) -> RegFile (200)
-> Control (100)
-> SignExtend and MUX (50)

Latency = 400 + 200 + 120 + 30 + 200 = 950ps = 980 - 30 (like sub but exclude the ALUSrc MUX like in lw)

# Additional Question 3

Why is the Control Unit not included in the critical path for Question 4 in the following way?



# Additional Answer 3

Why is the Control Unit not included in the critical path for Question 4 in the following way?

RegDst and WR are not required until the Register Write Back stage.

So the path from opcode -> Control -> Mux -> Registers is parallel to the path that generates WD.

For the same reason, we also do not consider the path opcode -> Control -> Register (RegWrite).

Earliest used result of the control unit is ALUSrc.