Week 7 Review

Question #0

Try these questions first:

1. Where are instructions stored?
2. How long is a single instruction?
3. What is the role of the Program Counter (PC)?
4. What do we mean by "instruction fetch"?
5. Where does the processor keep the instruction that is currently being executed?

Question #0

1. Where are instructions stored?
   - In memory, along with the input data values
2. How long is a single instruction?
   - 4 bytes (32 bits)
3. What is the role of the Program Counter (PC)?
   - Store the location (address) of the current instruction.
4. What do we mean by "instruction fetch"?
   - Retrieve an instruction from memory.
5. Where does the processor keep the instruction that is currently being executed?
   - In the Instruction Register.

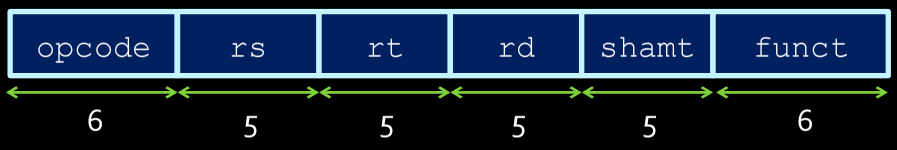
Question #1

- Your RAM unit has 6 address bits going into it. Given a 32-bit architecture, how many integers (words) is your RAM unit able to store?

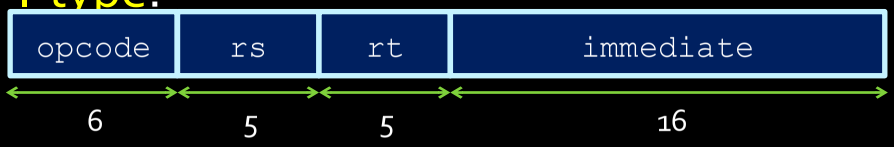
- Be careful here!
  - 6 address bits → 2^6 memory slots = 64 bytes.
  - 32-bit architecture → 4 bytes per integer.
  - RAM capacity = 64 / 4 = 16 integers in memory.

MIPS instruction types

- R-type:



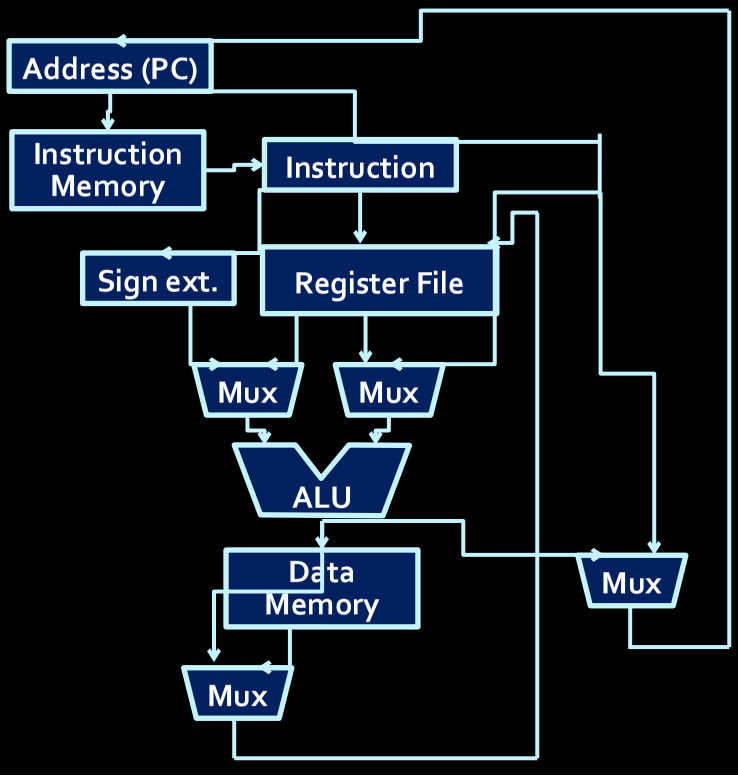
- I-type:



- J-type:

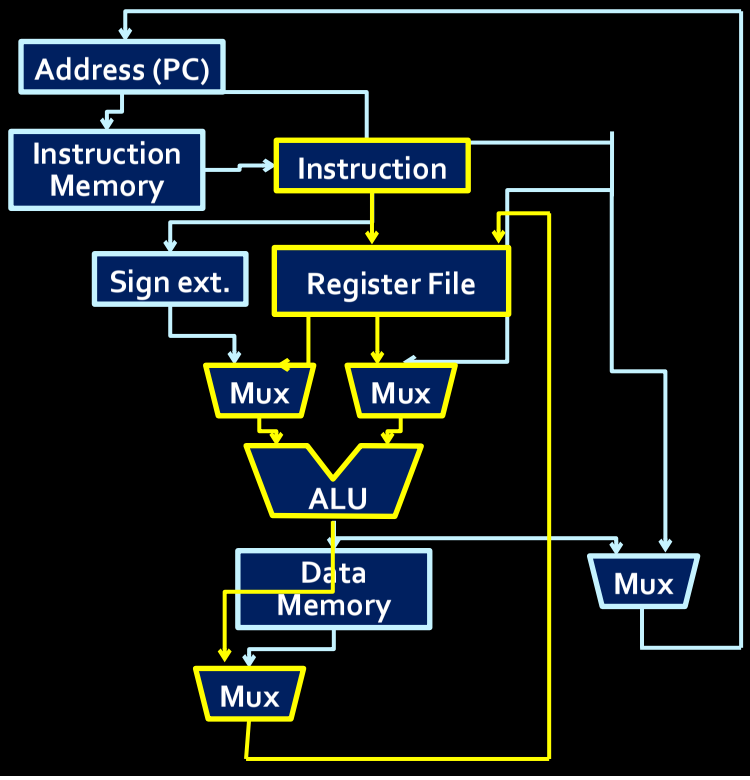
Question #2

What is the Datapath of an r-type instruction

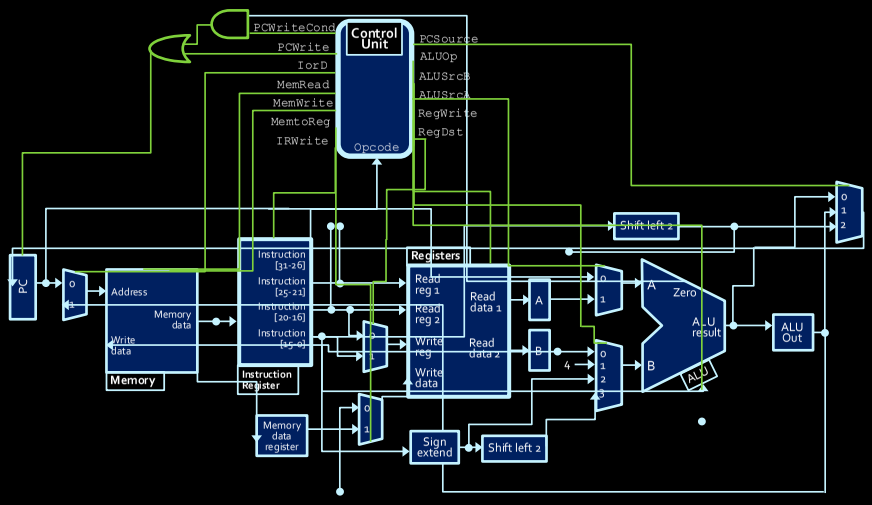


Question #2

What is the Datapath of an r-type instruction



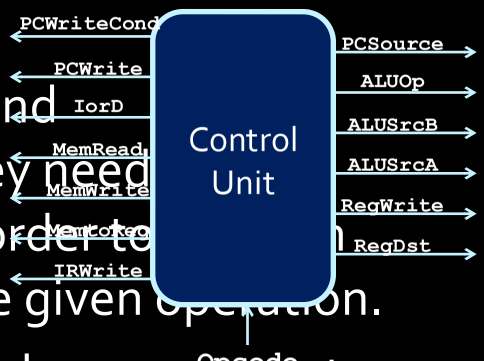
Question #3: Incrementing PC



- Given the datapath above, what signals would the control unit turn on and off to increment the program counter by 4? (PC ← PC + 4)

Controlling the signals

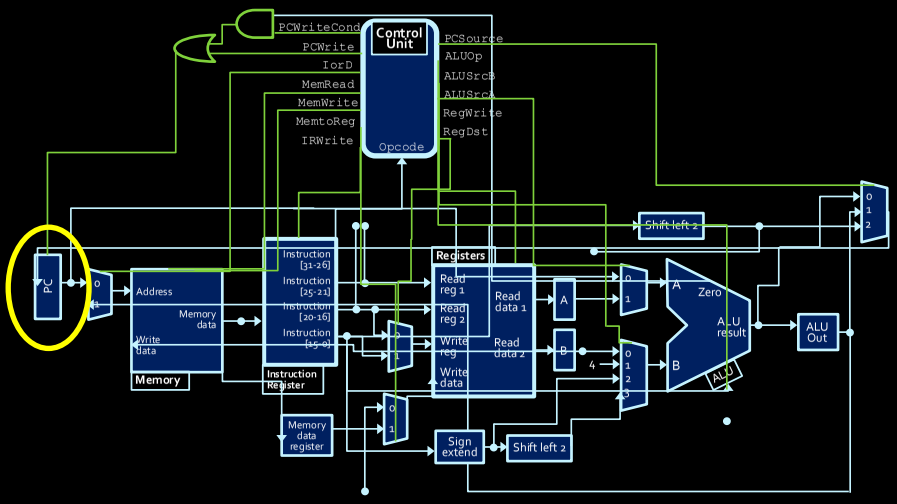
- Need to understand the role of each signal, and what value they need have in order to the given operation.
- So, what's the best approach to make this happen?



Basic approach to datapath

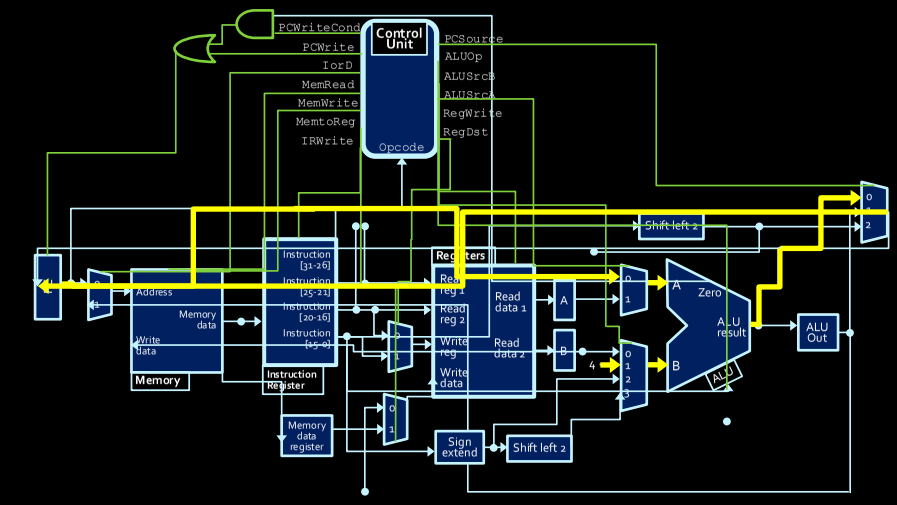
1. Figure out the data source(s) and destination.
2. Determine the path of the data.
3. Deduce the signal values that cause this path:
   a) Start with `Read` & `Write` signals
      - At most one ___`Write` signal should be high at a time.
   b) Then, mux signals along the data path.
   c) Non-essential signals get an `X` value.

Question #3: Incrementing PC



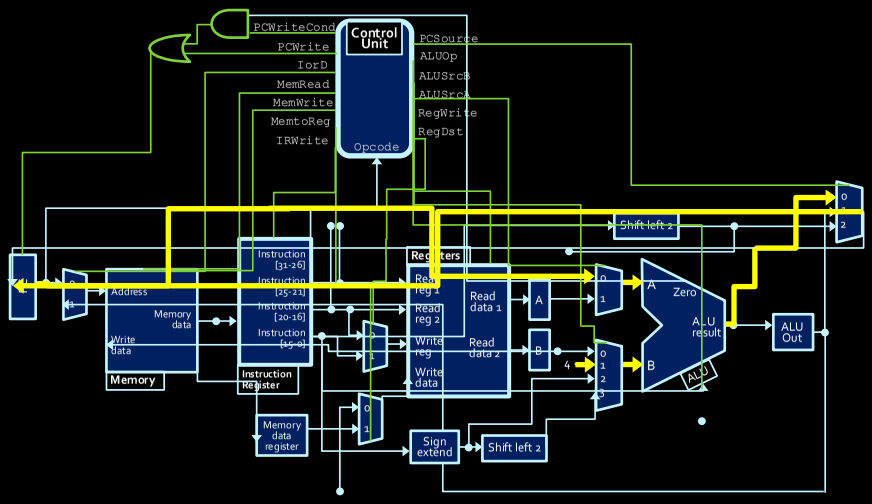
- Step #1: Determine data source and destination.
  - Program counter provides source,
  - Program counter is also destination.

Question #3: Incrementing PC



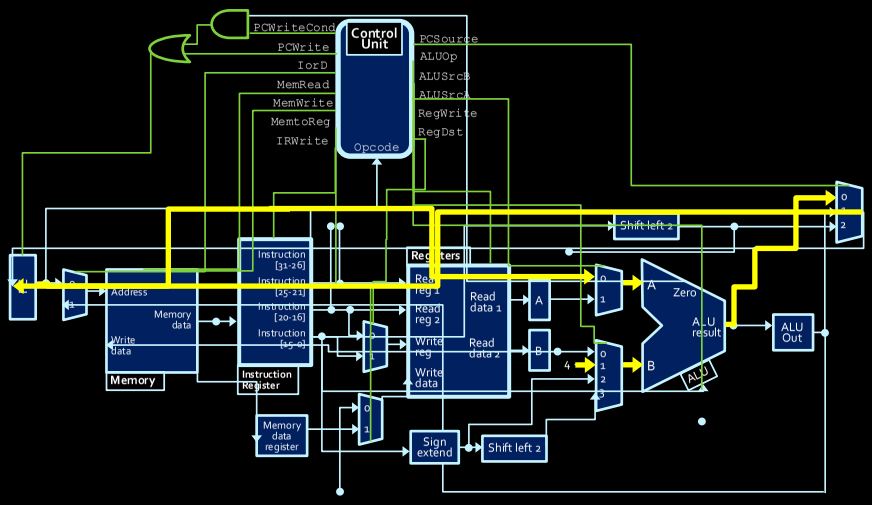
- Step #2: Determine path for data
  - Operand A for ALU: Program counter
  - Operand B for ALU: Literal value 4
  - Destination path: Through mux, back to PC

Question #3: Incrementing PC



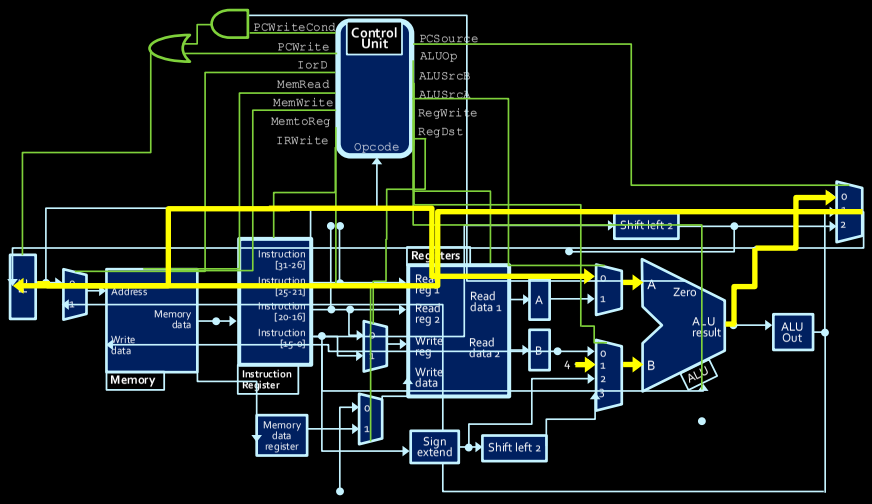
- Setting signals for this datapath:
  1. Read & Write signals:
     - `PCWrite` is high, all others are low.

Question #3: Incrementing PC



- Setting signals for this datapath:
  2. Mux signals:
     - `PCSource` is `0`, `AlUSrcA` is `0`, `ALUSrcB` is `1`
     - all others are "don't cares".

Question #3: Incrementing PC



- Other signals for this datapath:
  - `ALUOp` is `'ADD'`
  - `PCWriteCond` is `X` when `PCWrite` is `1`
    - Otherwise it is `0` except when branching.

Question #3 (final signals)

- `PCWrite = 1`
- `PCWriteCond = X`
- `IorD = X`
- `MemRead = 0`
- `MemWrite = 0`
- `MemToReg = X`
- `IRWrite = 0`
- `PCSource = 0`
- `ALUOp = 'ADD' (001)`
- `ALUSrcA = 0`
- `ALUSrcB = 01`
- `RegWrite = 0`
- `RegDst = X`

Question #4

0000 0000 0110 0101 0100 0000 0010 0111

- What is the type of this instruction?
- What does it do?
- Which register stores the result?

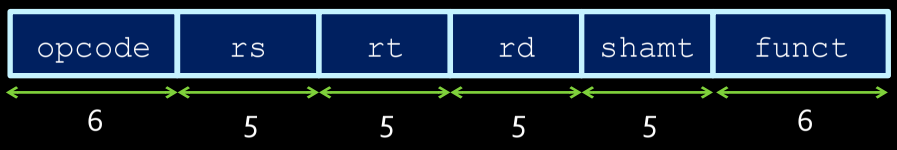
0000 0000 0110 0101 0100 0000 0010 0111

- What is the type of this instruction?
- What does it do?
- Which register stores the result?

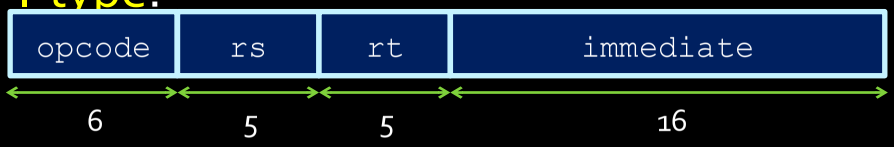
Instruction	Op/Func	Instruction	Op/Func
add	100000	srav	000111
addu	100001	srl	000010
addi	001000	srlv	000110
addiu	001001	beq	000100
div	011010	bgtz	000111
divu	011011	blez	000110
mult	011000	bne	000101
multu	011001	j	000010
sub	100010	jal	000011
subu	100011	jalr	001001
and	100100	jr	001000
andi	001100	lb	100000
nor	100111	lbu	100100
or	100101	lh	100001
ori	001101	lhu	100101
xor	100110	lw	100011
xori	001110	sb	101000
sll	000000	sh	101001
sllv	000100	sw	101011
sra	000011	mflo	010010

MIPS instruction types

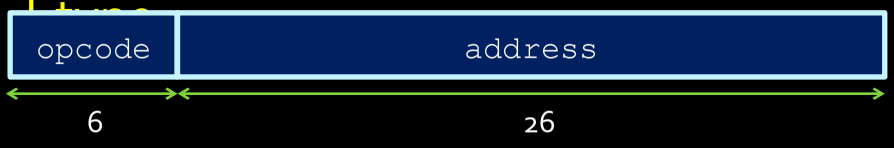
- R-type:



- I-type:



- J-type:



Question #4

0000 0000 0110 0101 0100 0000 0010 0111

- What is the type of this instruction?
- What does it do?
- Which register stores the result?

Question #4

0000 0000 0110 0101 0100 0000 0010 0111

- What is the type of this instruction?
  - R-type
- What does it do?
- Which register stores the result?

Question #4

0000 0000 0110 0101 0100 0000 0010 0111

- What is the type of this instruction?
  - R-type
- What does it do?
  - `funct = 100111`
- Which register stores the result?

```
0000 0000 0110 0101
0100 0000 0010 0111
```

funct = 100111

Instruction	Op/Func	Instruction	Op/Func
add	100000	srav	000111
addu	100001	srl	000010
addi	001000	srlv	000110
addiu	001001	beq	000100
div	011010	bgtz	000111
divu	011011	blez	000110
mult	011000	bne	000101
multu	011001	j	000010
sub	100010	jal	000011
subu	100011	jalr	001001
and	100100	jr	001000
andi	001100	lb	100000
nor	100111	lbu	100100
or	100101	lh	100001
ori	001101	lhu	100101
xor	100110	lw	100011
xori	001110	sb	101000
sll	000000	sh	101001
sllv	000100	sw	101011
sra	000011	mflo	010010

Question #4

0000 0000 0110 0101 0100 0000 0010 0111

- What is the type of this instruction?
  - R-type
- What does it do?
  - nor
- Which register stores the result?
  - `rd = 01000`
  - register 8 (known as `$t0` in MIPS assembly)

Question #5

Give the binary representation of the op-code to add 0x4027 (in decimal: 16423d) to the value of r3, and put the result in r5

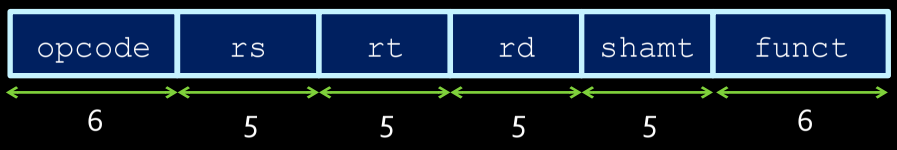
r5 ← r3 + 16423

0x4027

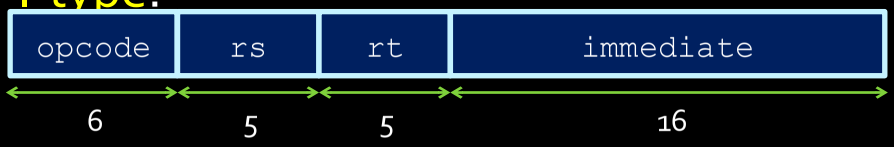
Instruction	Op/Func	Instruction	Op/Func
add	100000	srav	000111
addu	100001	srl	000010
addi	001000	srlv	000110
addiu	001001	beq	000100
div	011010	bgtz	000111
divu	011011	blez	000110
mult	011000	bne	000101
multu	011001	j	000010
sub	100010	jal	000011
subu	100011	jalr	001001
and	100100	jr	001000
andi	001100	lb	100000
nor	100111	lbu	100100
or	100101	lh	100001
ori	001101	lhu	100101
xor	100110	lw	100011
xori	001110	sb	101000
sll	000000	sh	101001
sllv	000100	sw	101011
sra	000011	mflo	010010

MIPS instruction types

- R-type:



- I-type:



- J-type:

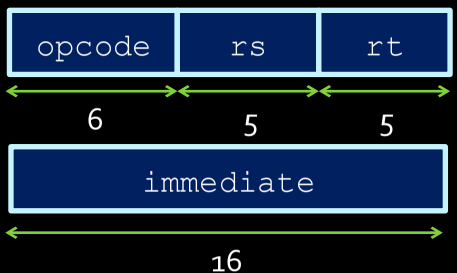
r5 ← r3 + 16423

0x4027

addi $5, $3, 16423

0x4027

00100000 01100101
01000000 00100111



Instruction	Op/Func	Instruction	Op/Func
add	100000	srav	000111
addu	100001	srl	000010
addi	001000	srlv	000110
addiu	001001	beq	000100
div	011010	bgtz	000111
divu	011011	blez	000110
mult	011000	bne	000101
multu	011001	j	000010
sub	100010	jal	000011
subu	100011	jalr	001001
and	100100	jr	001000
andi	001100	lb	100000
nor	100111	lbu	100100
or	100101	lh	100001
ori	001101	lhu	100101
xor	100110	lw	100011
xori	001110	sb	101000
sll	000000	sh	101001
sllv	000100	sw	101011
sra	000011	mflo	010010

Pay Attention!

- Very similar encodings produce difference results

```
                    addi $5, $3, 16423
00100000 01100101 01000000 00100111

00000000 01100101 01000000 00100111
                    nor $8, $3, $5
```