# September 24

- Test 1 review
- More programming
- Assignment 4 posted. Start EARLY.



# Question 1

[10] Assume that multiply instructions take 12 cycles and account for 10% of the instructions in a typical program and that the other 90% of the instructions require an average of 4 cycles for each instruction. What percentage of time does the CPU spend doing multiplication?

12 * 0.1 / (12*0.1 + 4*0.9) = 25%

## Question 2

[10] Consider a byte-addressable memory and an architecture that manipulates 2-byte words. What is the maximum number of words of memory available if the architecture has addresses that are 16 bits long?

2^16 bytes / (2 bytes/word) = 2^15 words = 32k



## Question 3

[10] A ZIP disk holds 100 megabytes. A typical MP3 music file requires 128 kbits per second. How many minutes of music can you store on a single disk?

100 * 2^20 bytes * 2^3 bits/byte / (2^17 bits/sec * 60 sec/min) = 100*2^6/60 = 106.7 minutes

## Question 4

1 [10] How many instructions that are 16 bits long fit in 32k bytes of memory?

32k / 2 = 16k



## Question 5

1 [10] Two machines have the same clock rate. What can you say about their performance relative to one another?

Nothing. They might have different ISA's or different CPI's.

# Question 6

1 [10] A certain program executes 200 million instructions. On a 300MHz Pentium II it takes 3 seconds to run. What is the MIPS rating of the processor on this program? What is the average CPI? On a 500MHz Pentium III the program takes 1 second. What is the MIPS rating for this processor? What is the CPI?

MIPS = 200/3 = 66.7, CPI = 300*3/200 = 4.5

MIPS = 200/1 = 200, CPI = 500*1/200 = 2.5



# Question 7

[10] Consider the characteristics of two machines M1 and M2. M1 has a clock rate of 400Mhz. M2 has a clock rate of 500MHz. There are 4 classes of instructions (A-D) in the instruction set. In a set of benchmark programs, the frequency of each class of instructions is shown in the table. How many millions of instructions per second (MIPS) does each machine execute on average?

| Instruction Class | Frequency | M1 CPI | M2 CPI |
|---|---|---|---|
| A | 40% | 2 | 2 |
| B | 30% | 3 | 2 |
| C | 20% | 4 | 3 |
| D | 10% | 5 | 4 |

M1 MIPS = 400 / (0.4*2 + 0.3*3 + 0.2*4 + 0.1*5) = 133.3

M2 MIPS = 500 / (0.4*2 + 0.3*2 + 0.2*3 + 0.1*4) = 208.3

## Question 8

1 [10] Which of the machines above is faster on average? By what factor?

M2 is faster by a factor of 1.6



## Question 9

[10] In a certain set of benchmark programs about every 4th instruction is a load instruction that fetches data from main memory. The time required for a load is 50ns. The CPI for all other instructions is 4. Assuming the ISA's are the same, how much faster will the benchmarks run with a 1GHz clock than with a 500MHz clock?

For 4 instructions the time is 50ns + 3*4/R, so the speedup is (50ns + 24ns) / (50ns + 12ns) = 1.19 or about 19%.

# Question 10

1.[10] State Amdahl's Law with an equation.

Timproved = (Tunaffected + Taffected/improvement)



# Grade Distribution

10~19: 1
20~29: 0
30~39: 2
40~49: 2
50~59: 3
60~69: 4
70~79: 6
80~89: 13
90~99: 12
100: 2

**READ THE BOOK!**

# So far we've learned:

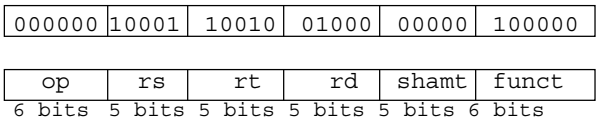
- MIPS
  - — loading words but addressing bytes
  - — arithmetic on registers only

| Instruction | Meaning |
|---|---|
| `add $s1, $s2, $s3` | `$s1 = $s2 + $s3` |
| `sub $s1, $s2, $s3` | `$s1 = $s2 – $s3` |
| `lw $s1, 100($s2)` | `$s1 = Memory[$s2+100]` |
| `sw $s1, 100($s2)` | `Memory[$s2+100] = $s1` |



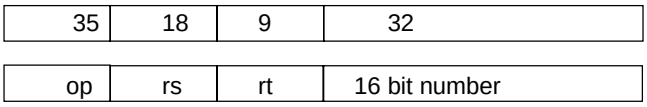
# Machine Language

- Instructions, like registers and words of data, are also 32 bits long
  - Example: `add $t0, $s1, $s2`
  - registers have numbers, `$t0=8, $s1=17, $s2=18`
- Instruction Format:



# Machine Language

- Consider the load-word and store-word instructions,
  - What would the regularity principle have us do?
  - New principle: **Good design demands a compromise**
- Introduce a new type of instruction format
  - I-type for data transfer instructions
  - other format was R-type for register
- Example: `lw $t0, 32($s2)`

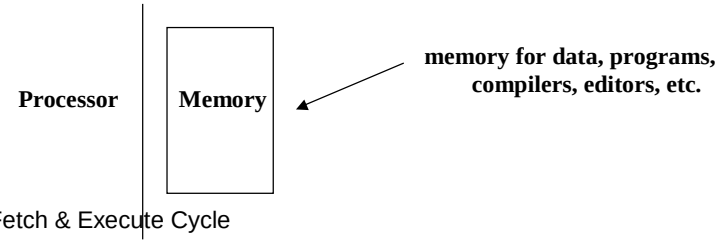



- Where's the compromise?



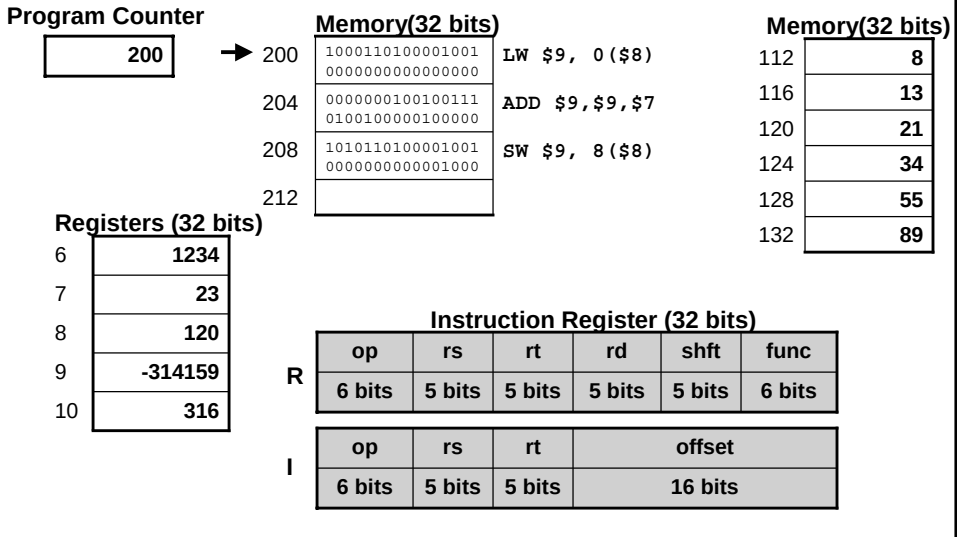
# Stored Program Concept

- Instructions are bits
- Programs are stored in memory
  — to be read or written just like data



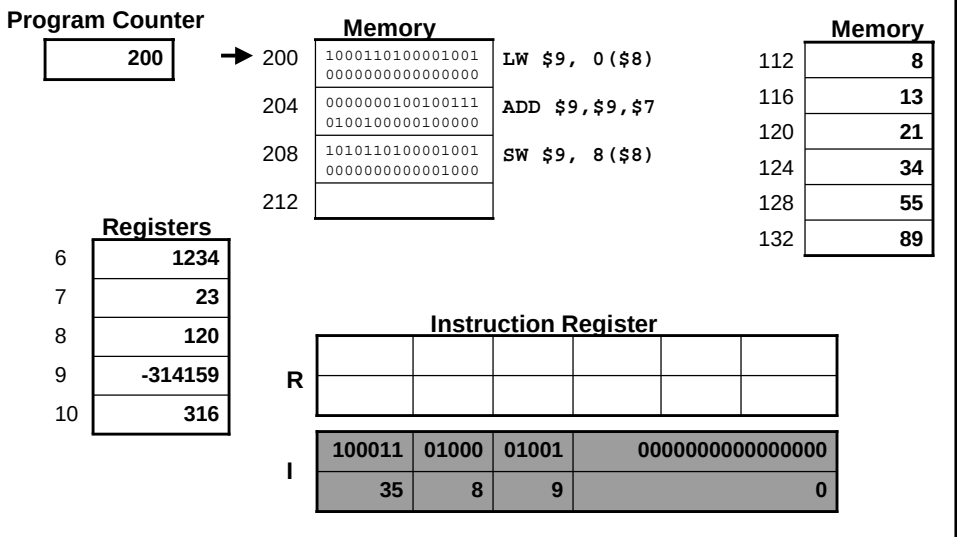

- Fetch & Execute Cycle
  - Instructions are fetched and put into a special register
  - Bits in the register "control" the subsequent actions
  - Fetch the “next” instruction and continue

# Execution Example

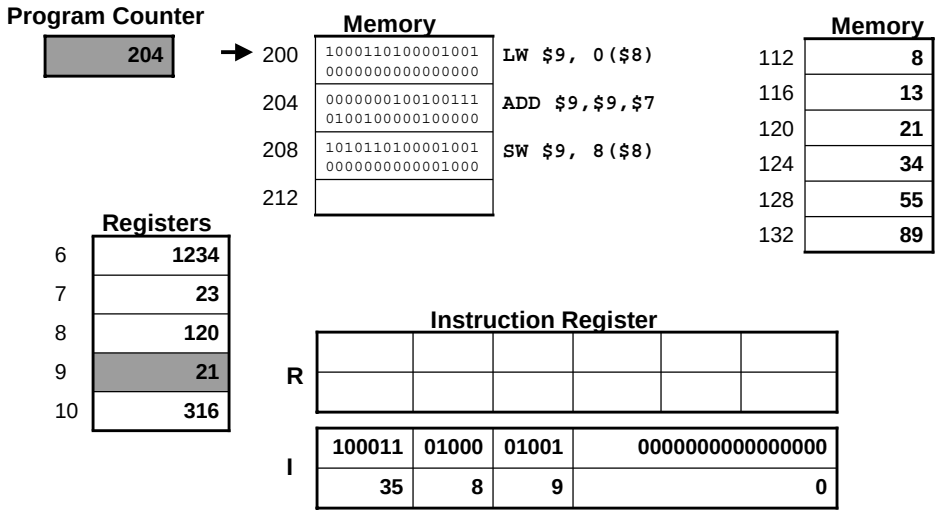





# Execution Example: Fetch(200)

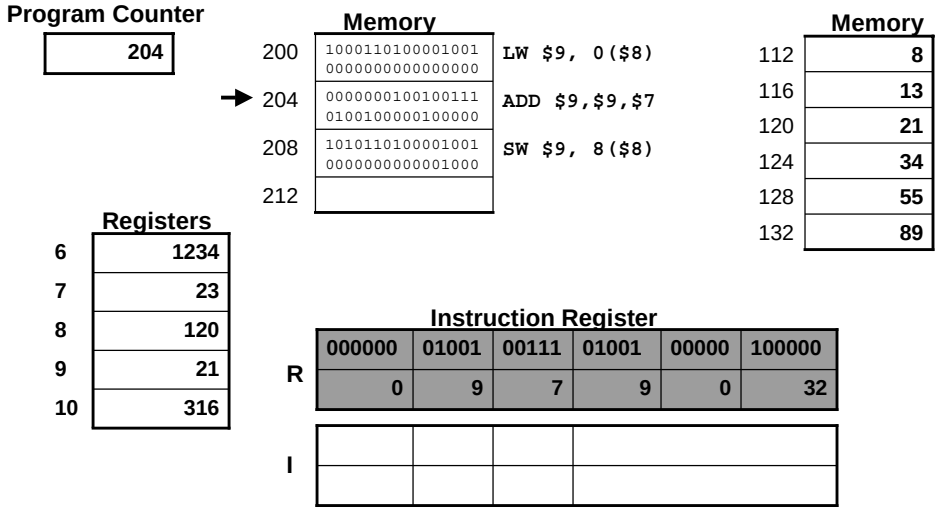


# Execution Example: Execute(200)

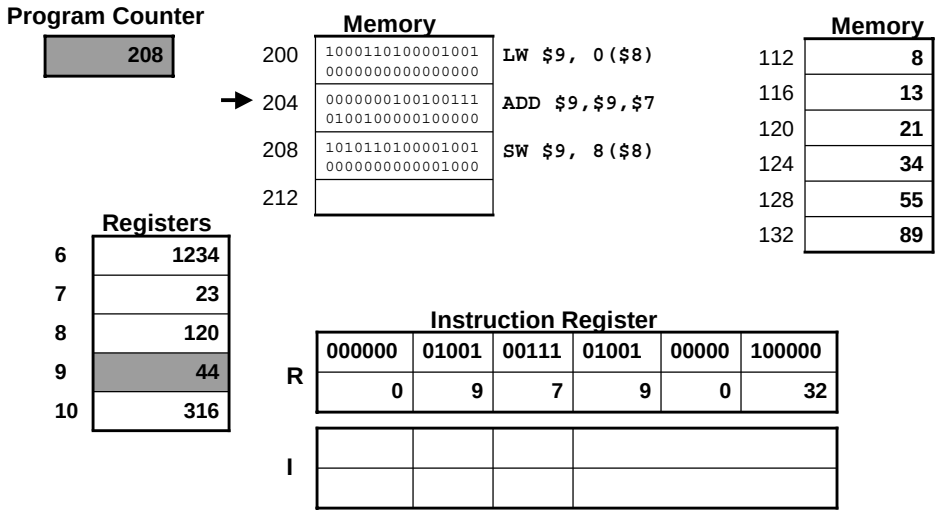





# Execution Example: Fetch(204)

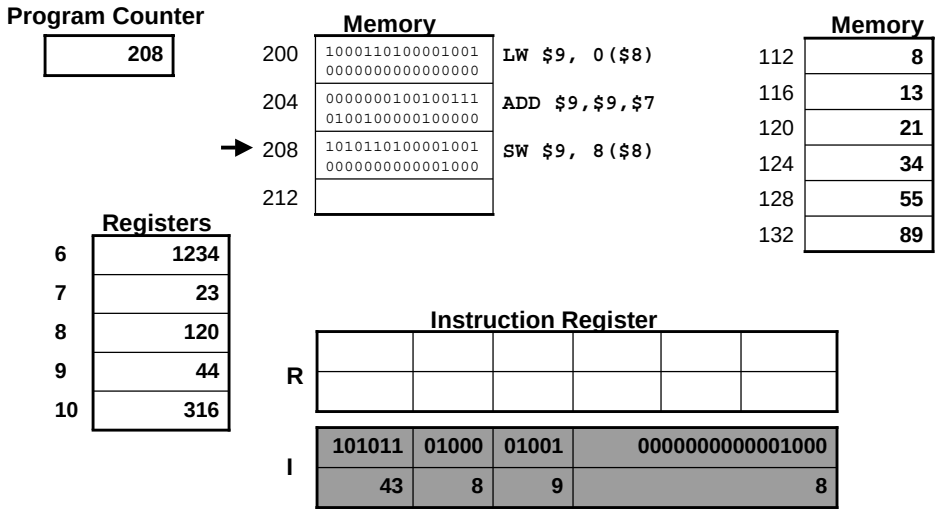


# Execution Example: Execute(204)

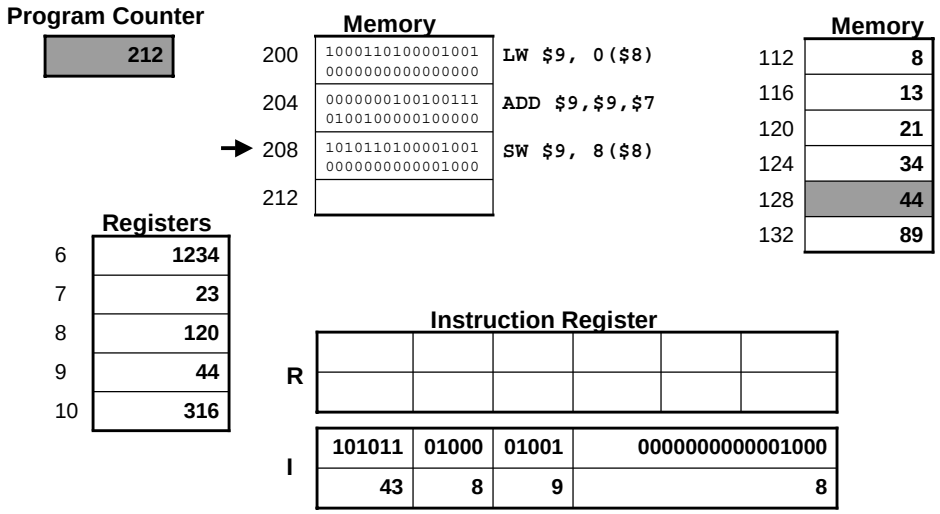





# Execution Example: Fetch(208)



# Execution Example: Execute(208)






# Control

- Decision making instructions
  - alter the control flow,
  - i.e., change the "next" instruction to be executed
- MIPS conditional branch instructions:

```
bne $t0, $t1, Label
beq $t0, $t1, Label
```

- Example: if (i==j) h = i + j;

```
        bne $s0, $s1, Label
        add $s3, $s0, $s1
Label: ....
```

# Control

- MIPS unconditional branch instructions:

```
j  label
```

- Example:

```
if (i!=j)            beq $s4, $s5, Lab1
    h=i+j;           add $s3, $s4, $s5
else                 j Lab2
    h=i-j;           Lab1:  sub $s3, $s4, $s5
                     Lab2:  ...
```

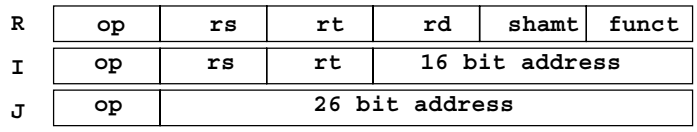


# So far:

- Instruction Meaning

```
add $s1,$s2,$s3   $s1 = $s2 + $s3
sub $s1,$s2,$s3   $s1 = $s2 – $s3
lw $s1,100($s2)   $s1 = Memory[$s2+100]
sw $s1,100($s2)   Memory[$s2+100] = $s1
bne $s4,$s5,L     Next instr. is at Label if $s4 != $s5
beq $s4,$s5,L     Next instr. is at Label if $s4 = $s5
j Label           Next instr. is at Label
```

- Formats:



# Control Flow

- We have: beq, bne, what about Branch-if-less-than?
- New instruction:

```
                                  if  $s1 < $s2 then
                                     $t0 = 1
      slt $t0, $s1, $s2           else
                                     $t0 = 0
```

- Can use this instruction to build "`blt $s1, $s2, Label`"
  — can now build general control structures
- Note that the assembler needs a register to do this,
  — there are policy of use conventions for registers



# Policy of Use Conventions

| Name | Register number | Usage |
|---|---|---|
| `$zero` | 0 | the constant value 0 |
| `$v0-$v1` | 2-3 | values for results and expression evaluation |
| `$a0-$a3` | 4-7 | arguments |
| `$t0-$t7` | 8-15 | temporaries |
| `$s0-$s7` | 16-23 | saved |
| `$t8-$t9` | 24-25 | more temporaries |
| `$gp` | 28 | global pointer |
| `$sp` | 29 | stack pointer |
| `$fp` | 30 | frame pointer |
| `$ra` | 31 | return address |