September 17

- Test 1 pre(re)view
- Fang-Yi will demonstrate Spim

2

2. [10] Consider a byte-addressable memory and an architecture that manipulates 4-byte words. What is the maximum number of words of memory available if the architecture has 13 bit addresses?

2^13 bytes / (2^2 bytes/word) = 2^11 words = 2048

3

3. [5] How many bits are in a megabyte?

2^20 bytes/megabyte * 2^3 bits/byte = 2^23 bits/megabyte
= 8 megabits = 8,388,608

4

4. [3] How many MIPS instructions fit in 32k bytes of memory?

32 = 2^5
k = 2^10
32k = 2^5 * 2^10 = 2^15
MIPS instructions are 4 bytes = 2^2

2^15 / 2^2 = 2^13 instructions = 8k = 8192

5

5. [12] Give as many different formulas as you can for execution time using the following variables. Each equation should be minimal; that is, it should not contain any variable that is not needed. CPI, R=clock rate, T=cycle time, M=MIPS, I=number of instructions in program, C=number of cycles in program.

(CPI * I * T)
(CPI * I / R)
(I / (M * 10^6))
(C * T)
(C / R)
(C / (CPI * M * 10^6))

6

6. [10] Consider the characteristics of two machines M1 and M2. M1 has a clock rate of 500Mhz. M2 has a clock rate of 600MHz. 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.

Instruction Class	Frequency	M1 CPI	M2 CPI
A	40%	2	2
B	25%	3	2
C	25%	3	3
D	10%	5	4

What is the average CPI for each machine?

M1 = 0.4*2 + 0.25*3 + 0.25*3 + 0.1*5 = 2.8
M2 = 0.4*2 + 0.25*2 + 0.25*3 + 0.1*4 = 2.45

7

7. [10] How much faster is M2 than M1?

(2.8 / 500) / (2.45 / 600) = 1.37

37% faster.

8

8. [5] What is the cycle time of each machine?

M1 = 1/500MHz = 2ns = 2*10^-9 s
M2 = 1/600MHz = 1.67 ns = 1.67 * 10^-9 s

9

9. [10] If we make a low power version of M2 with a lower clock rate, what clock rate will we need to match the performance of M1?

2.8 / 500 = 2.45 / X

X = 437.5MHz

10

10. [15] You are going to enhance a machine, and there are two possible improvements: either make multiply instructions run 4 times faster than before, or make memory access instructions run 2 times faster than before. You repeatedly run a program that takes 100 seconds to execute. Of this time, 20% is used for multiplication, 50% for memory access instructions, and 30% for other tasks. What will be the speedup if you improve only multiplication? What will be the speedup if you improve only memory access? What will be the speedup if both improvements are made?

Speedup if we improve only multiplication = 100 / (30 + 50 + 20/4) = 100/85 = 1.18

Speedup if we only improve memory access = 100 / (30 + 50/2 + 20) = 100/75 = 1.33

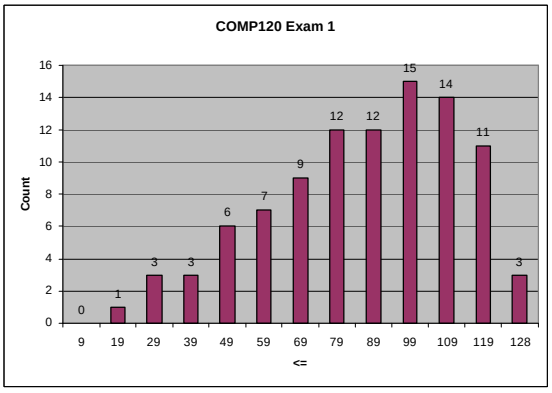
Speedup if both improvements are made = 100 / (30 + 50/2 + 20/4) = 100/60 =1.67

12

12. [3] How many bytes of memory are used by an array of 1000 pointers on the MIPS architecture?

1000 pointers * 32 bits/address / 8 bits/byte = 4000 bytes

Grade Distribution



SPIM Demonstration

Chapter 3

Programming the Machine

Instructions:

- Language of the Machine
- More primitive than higher level languages
  e.g., no sophisticated control flow
- Very restrictive
  e.g., MIPS Arithmetic Instructions

- We'll be working with the MIPS instruction set architecture
  - similar to other architectures developed since the 1980's
  - used by NEC, Nintendo, Silicon Graphics, Sony

Design goals: maximize performance and minimize cost, reduce design time

MIPS arithmetic

- All instructions have 3 operands
- Operand order is fixed (destination first)

Example:

C code: `A = B + C`

MIPS code: `add $s0, $s1, $s2`

(associated with variables by compiler)

MIPS arithmetic

- Design Principle: **simplicity favors regularity**. Why?
- Of course this complicates some things...

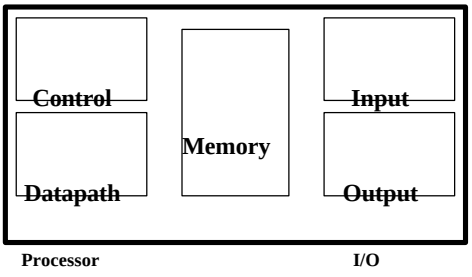
C code:
```
A = B + C + D;
E = F - A;
```

MIPS code:
```
add $t0, $s1, $s2
add $s0, $t0, $s3
sub $s4, $s5, $s0
```

- Operands must be registers, only 32 registers provided
- Design Principle: **smaller is faster**. Why?

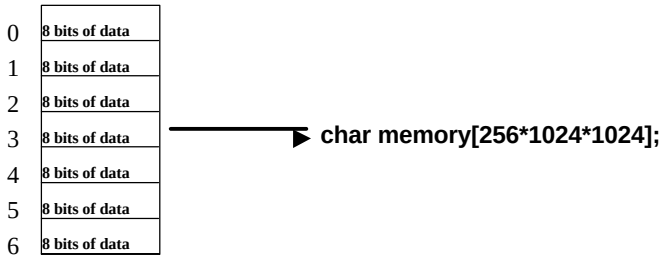
Registers vs. Memory

- Arithmetic instructions operands must be registers,
  — only 32 registers provided
- Compiler associates variables with registers
- What about programs with lots of variables?



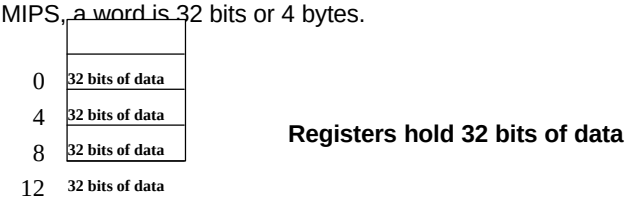
Memory Organization

- Viewed as a large, single-dimension array, with an address.
- A memory address is an index into the array
- "Byte addressing" means that the index points to a byte of memory.



Memory Organization

- Bytes are nice, but most data items use larger "words"
- For MIPS, a word is 32 bits or 4 bytes.



- 2^{32} bytes with byte addresses from 0 to $2^{32}-1$
- 2^{30} words with byte addresses 0, 4, 8, ... $2^{32}-4$
- Words are aligned
  i.e., what are the least 2 significant bits of a word address?

Endians?

- What order are the bytes inside the word?
- Is byte 0 the high-order bits of word 0?
- Or is it the low order bits?

- "Big Endian" byte 0 is high-order bits [0 1 2 3]
  - Macintosh, SPARC
- "Little Endian" byte 0 is low-order bits [3 2 1 0]
  - Intel, DECStation

When would I care?

Instructions

- Load and store instructions
- Example:

C code: `A[8] = h + A[8];`

MIPS code:
```
lw $t0, 32($s3)
add $t0, $s2, $t0
sw $t0, 32($s3)
```

- Store word has destination last
- Remember arithmetic operands are registers, not memory!

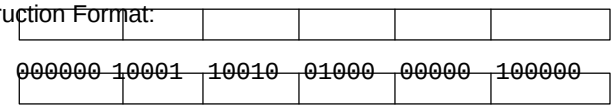
So far we've learned:

- 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

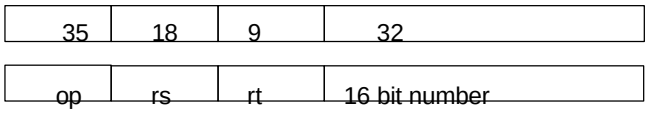
- Instructions, like registers and words of data, are also 32 bits long
  - Example: `add $t0, $s1, $s2`
  - registers have numbers, `$t0=8, $s1=17, $s2=18`
- Instruction Format:



op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Machine Language

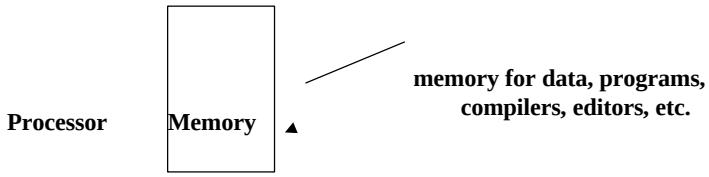
- Consider the load-word and store-word instructions,
  - What would the regularity principle have us do?
  - New principle: **Good design demands a compromise**
- Introduce a new type of instruction format
  - I-type for data transfer instructions
  - other format was R-type for register
- Example: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