CS 2630
Computer Organization

Meeting 4: Intro to Architecture (specifically, MIPS)
Brandon Myers
University of Iowa
Where we are going

Compiler
- translating source code (C or Java)
- Programs to assembly language
- And linking your code to Library code

Instruction set architecture (e.g., MIPS)
- How the software talks to the hardware

Memory system

Processor
- How a processor runs MIPS
- Programs!

I/O system
- How switches (1 or 0) can be used to build Interesting functions:
  - from integer arithmetic to programmable computers

Datapath & Control

Digital logic
Where we are going

NEXT what gets stored in here

Learned some bitwise operations

Learned how bits are organized in memory
Where we are going

- Compiler
  - translating source code (C or Java)
  - Programs to assembly language
  - And linking your code to Library code

- Instruction set architecture (e.g., MIPS)
  - How the software talks to the hardware

- Memory system
- Processor
- I/O system

- Datapath & Control
  - How a processor runs MIPS Programs!

- Digital logic
  - How switches (1 or 0) can be used to build Interesting functions:
    - from integer arithmetic to programmable computers
Representations of a program

```
int x = arr[1];
arr[2] = x + 10;
```

Compiler

```
lw $t0, 4($r0)
addi $t0, $t0, 10
sw $t0, 8($r0)
```

Assembler

```
10001110000010000000000000000100
001000100010000000000000000001010
1010111000001000000000000000000100
```

High level language program (human readable)

assembly program as text (CS2630 student readable)

assembly program as binary (machine readable)
Languages

- High level language (HLL) programs are instruction set architecture-independent
  - language is convenient to use
  - language is has powerful features

- Assembly language programs are specific to an instruction set architecture
  - the “native language” of the processor
  - language is bare-bones: lacks most of the features of HLLs

in this course, these are all synonyms
- Instruction set architecture
- instruction set
- ISA
- architecture

Java, C, Python, Lisp, C++, MATLAB, ...

MIPS, ARM, x86, SPARC, Power, Alpha, ...
Brief history of compatibility

before IBM 360 – different architecture for every processor

- Processor model 1: slow processor, small memory
- Processor model 2: mediocre processor, medium memory
- Processor model 3: fast processor, large memory

my_program.asm version 1
my_program.asm version 2
my_program.asm version 3

post IBM 360 – one architecture for all processors

- my_program.asm

one program written for one architecture, IBM 360

runs on different microarchitectures

- IBM Model 30: slow processor, small memory
- IBM Model 50: mediocre processor, medium memory
- IBM Model 75: fast processor, large memory
Peer instruction

I wrote my program in C (a high level language) and compiled it to run on a MIPS machine. What should I do to get that program to run on an x86 machine?

a) rewrite my C program to be compatible with x86
b) rewrite my program in x86 assembly language
c) recompile my C program for x86
d) reassemble my program using the x86 assembler
Languages

• High level language: usually has variables, objects, and arrays

• Assembly language: usually just has registers and memory
The main idea

- 4 categories of *instructions*
  - perform an operation on two registers and store result in a register
  - perform an operation on one register and a constant and store the result in a register
  - move a value between a register and memory
  - determine which instruction to execute next
Register transfer language

- 4 categories of *instructions*
  - perform an operation on two registers and store result in a register
  - perform an operation on one register and a constant and store the result in a register
  - move a value between a register and memory
  - determine which instruction to execute next

```plaintext
if r0 == r1
    PC <- branchAddress
else
    PC <- PC+4
```

- `r0 <- M[address]`
- `M[address] <- r0`
- `r1 <- r0 + constant`
- `r2 <- r1 + r0`
Example assembly program

**MEMORY**

<table>
<thead>
<tr>
<th>address</th>
<th>0x0</th>
<th>0x4</th>
<th>0x8</th>
<th>0xC</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>-14</td>
<td>50</td>
<td>1</td>
</tr>
</tbody>
</table>

**REGISTERS**

- r0
- r1
- r2

**high level**

\[ a = x + y - z \]

**Peer instruction:**
Write an assembly program using a sequence of instructions (use register transfer language)
In groups of 3: All of these questions are answerable from the information on the sheet.

1. What does the slt instruction do? Be specific about what happens to each operand.
2. What is the opcode (numeric value) for the subu instruction?
3. What does the notation \{imm, 00\} mean?
5. What is the difference between sb and sw, in terms of what the instructions do? (not just asking about the notes column)
6. Find the instruction you would use to put an immediate in the most significant 16-bits of a given register and 0’s in the least significant 16 bits.
Administrivia

• Monday Labor day
• Reschedule only next week’s Debug Your Brain to 9/5, 2-3pm, 201K MLH
• Wednesday is Lab 2, Pre-lab posted soon
• HW2 due date changed to 9/11