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

I/O system
- How a processor runs MIPS
  Programs!

Datapath & Control
- How switches (1 or 0) can be used
  to build Interesting functions:
  from integer arithmetic to
  programmable computers

Digital logic
Where we are going

- Instruction memory
- Execution engine
- Data memory

Learned some bitwise operations
Learned how bits are organized in memory

NEXT what gets stored in here
Where we are going

- Instruction set architecture (e.g., MIPS)
- Compiler: translating source code (C or Java) to assembly language.
- Memory system
- Processor
- I/O system
- Datapath & Control
- Digital logic

How the software talks to the hardware:
- How a processor runs MIPS programs!
- How switches (1 or 0) can be used to build interesting functions: from integer arithmetic to programmable computers.
Reading quiz

1. In MIPS, what do load word (lw) and store word (sw) instructions do?

2. What is the name for constants in MIPS instructions?

3. What does this instruction do?
   add $t0, $t1, $t3

4. What is the difference between memory and registers?
Representations of a program

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

High level language program (human readable)

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

assembly program as text (CS2630 student readable)

```
10001110000010000000000000000100
00100010001000000000000000001010
10101110000010000000000000001000
```

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
  
  
  \[
  \text{Java, C, Python, Lisp, C++, MATLAB, ...}
  \]

• 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

\[
\text{MIPS, ARM, x86, SPARC, Power, Alpha, ...}
\]
Brief history of compatibility

before IBM 360 – different architecture for every processor

- my_program.asm version 1
  Processor model 1
  slow processor, small memory

- my_program.asm version 2
  Processor model 2
  mediocre processor, medium memory

- my_program.asm version 3
  Processor model 3
  fast processor, large memory

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 MINS 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
Microprocessor Transistor Counts 1971

Date of introduction

https://library.law.uiowa.edu/history-library
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

<table>
<thead>
<tr>
<th>MEMORY</th>
<th>REGISTERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>r0</td>
</tr>
<tr>
<td>0x0</td>
<td>0</td>
</tr>
<tr>
<td>0x4</td>
<td>-14</td>
</tr>
<tr>
<td>0x8</td>
<td>50</td>
</tr>
<tr>
<td>0xC</td>
<td>1</td>
</tr>
</tbody>
</table>

$a = x + y - z$

Peer instruction:
Write an assembly program using a sequence of instructions (use register transfer language)
MIPS reference sheet scavenger hunt

In groups of 3: All of these questions are answerable from the information on the sheet.

1. What does the sll 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?
4. What does the andi instruction do? Be specific about what happens to each operand.
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.