A computer that plays cards

1. Come up with a binary encoding for a 52-card deck

Operations on a pair of cards

2. We want the following operations to be easy to implement
   • Compare two cards, which is higher value?
   • Compare two cards, are they the same suit?
If you are still trying to get into the class

• Consider using an add/drop form to sign up for ECE 3350 Computer Architecture and Organization
  • equivalent to CS2630 in terms of content and program requirements/pre-requisites
CS 2630
Computer Organization

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

Instruction set architecture (e.g., MIPS)

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

Instruction set architecture (e.g., MIPS)

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
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)
- Programs to assembly language
- And linking your code to Library code

How the software talks to the hardware

Memory system

Processor

I/O system

Datapath & Control

Digital logic

How a processor runs MIPS Programs!

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
001000010000100000000000000001010
101011100000100000000000000001000
```

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

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
Processor 1
slow processor, small memory
Processor 2
mediocre processor, medium memory
Processor 3
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

![Diagram](image.png)
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

(later!)
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>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></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)
What to do next

• HW1/Quiz1 due tonight
• Readings up on architecture and MIPS
• HW2/Quiz2 up by tomorrow, due Feb 2
  • Install MARS on the resources page of the website, you’ll use it in HW2
  • Watch MARS videos 1&2 on the Modules page
• come to debug your brain tonight