Programming in HMMM

BBM103 Introduction to Programming Lab 1
Week 3

Fall 2017
Von Neumann Architecture

• A **program** (a list of instructions) is stored in the main memory.
  - **Stored Program Concept**

• Instructions are copied (one at a time) into the **instruction register** in the CPU for execution.
Von Neumann Architecture

Programs are stored in memory in **machine language**.
The Power of the Stored Program

- A program written in machine language is a series of binary numbers representing the instructions stored in memory.

- The *stored program* concept is a key reason why computers are so powerful:
  - Running a different program does not require large amounts of time and effort to reconfigure or rewire hardware; it only requires writing the new program to memory.
Assembly Language

• **Assembly language** is a human-readable machine language.

• Instead of programming in binary (0’s and 1’s), it is easier to use an assembly language.

• An **assembler** is a computer program that interprets software programs written in assembly language into machine language.

```
0000 0001 0000 0001
1000 0010 0001 0001
0110 0010 0010 0001
0000 0010 0000 0010
0000 0000 0000 0000
```

```
0110 0010
read r1

0000 0001
mul r2 r1 r1

0000 0001
add r2 r2 r1

write r2

halt
```

"mnemonics" instead of bits
The Harvey Mudd Miniature Machine (HMMM)

• Hmm (Harvey Mudd Miniature Machine) is a 16-bit, 23-instruction simulated assembly language with \(2^8=256\) 16-bit words of memory.

• In addition to the **program counter** and **instruction register**, there are 16 registers named \(r_0\) through \(r_{15}\).

<table>
<thead>
<tr>
<th>Hmm assembly code</th>
<th>Corresponding instructions in machine language</th>
</tr>
</thead>
<tbody>
<tr>
<td>0     read       r1</td>
<td>0000 0001 0000 0001</td>
</tr>
<tr>
<td>1     read       r2</td>
<td>0000 0010 0000 0001</td>
</tr>
<tr>
<td>2     mul        r1 r1 r2</td>
<td>1000 0001 0001 0010</td>
</tr>
<tr>
<td>3     setn       r2 2</td>
<td>0001 0010 0000 0010</td>
</tr>
<tr>
<td>4     div        r1 r1 r2</td>
<td>1001 0001 0001 0010</td>
</tr>
<tr>
<td>5     write      r1</td>
<td>0000 0001 0000 0010</td>
</tr>
<tr>
<td>6     halt</td>
<td>0000 0000 0000 0000</td>
</tr>
</tbody>
</table>
The Harvey Mudd Miniature Machine (HMMM)

Central processing unit registers

General-purpose register r1

General-purpose register r2

16 registers

Random access memory locations

Von Neumann bottleneck

CPU

0

read r1

1

mul r2 r1 r1 r1

2

add r2 r2

3

write r2

4

halt

256 memory locations
The Harvey Mudd Miniature Machine (HMMM)

**read r1**
reads from keyboard into reg1

**write r2**
outputs reg2 onto the screen

**setn r1 42**
reg1 = 42

**addn r1 -1**
reg1 = reg1 - 1

**add r3 r1 r2**
reg3 = reg1 + reg2

**sub r3 r1 r2**
reg3 = reg1 - reg2

**mul r2 r1 r1**
reg2 = reg1 * reg1

**div r1 r1 r2**
reg1 = reg1 / reg2

**integers only!**

**reads**

**writes**

**sets**

**adds**

**subtracts**

**multiplies**

**divides**

**you can replace 42 with anything from -128 to 127**

**a shortcut**

read r1
write r2

setn r1 42
addn r1 -1
add r3 r1 r2
sub r3 r1 r2
mul r2 r1 r1
div r1 r1 r2

reg1 = 42
reg1 = reg1 - 1
reg3 = reg1 + reg2
reg3 = reg1 - reg2
reg2 = reg1 * reg1
reg1 = reg1 / reg2

you can replace 42 with anything from -128 to 127

**integers only!**
The Harvey Mudd Miniature Machine (HMMM)

System instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>halt</td>
<td>Stop!</td>
<td></td>
</tr>
<tr>
<td>read rX</td>
<td>Place user input in register rX</td>
<td></td>
</tr>
<tr>
<td>write rX</td>
<td>Print contents of register rX</td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td>Do nothing</td>
<td></td>
</tr>
</tbody>
</table>

Setting register data

| setn rX N  | Set register rX equal to the integer N (-128 to +127) |         |
| addn rX N  | Add integer N (-128 to +127) to register rX |         |
| copy rX rY | Set rX = rY | mov |

Arithmetic

| add rX rY rZ | Set rX = rY + rZ |         |
| sub rX rY rZ | Set rX = rY - rZ |         |
| neg rX      | Set rX = -rX |         |
| mul rX rY rZ | Set rX = rY * rZ |         |
| div rX rY rZ | Set rX = rY / rZ (integer division; no remainder) |         |
| mod rX rY rZ | Set rX = rY % rZ (returns the remainder of integer division) |         |

Jumps!

| jump N      | Set program counter to address N |         |
| jump rX     | Set program counter to address in rX |         |
| begin rX N  | If rX = 0, then jump to line N | negz |
| iseq rX N   | If rX = 0, then jump to line N | meqz |
| isp rX N    | If rX > 0, then jump to line N | negz |
| jslt rX N   | If rX < 0, then jump to line N | jslt |
| call rX     | Copy the next address into rX and then jump to mem. addr. N | call |

Interacting with memory (RAM)

| loadm rX N  | Load register rX with the contents of memory address N |         |
| storem rX N | Store contents of register rX into memory address N |         |
| loadr rX rY | Load register rX with data from the address location held in reg. rY | load, load |
| storer rX rY | Store contents of register rX into memory address held in reg. rY | storel, store |

Hmmm

the complete reference

At

www.cs.hmc.edu/~cs5grad/cs5/hmmm/documentation/documentation.html
Example #1:

What does this program do?

CPU

Central processing unit *registers*

**r1**
General-purpose register r1

**r2**
General-purpose register r2

RAM

Random access memory locations

0
**read r1**

1
**mul r2 r1 r1**

2
**add r2 r2 r1**

3
**write r2**

4
**halt**

Screen

6 (input)
Example #1 (cont.):

Screen

6 (input)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>read r1</td>
<td># Get input from user to r1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>mul r2 r1 r1</td>
<td># r2 = r1 * r1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>add r2 r2 r1</td>
<td># r2 = r2 + r1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>write r2</td>
<td># Print the contents of register r2 on standard output</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>halt</td>
<td># Halt program</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Jumps in HMMM

\[
\begin{align*}
\text{jeqzn } r1 & \ 42 \quad \text{IF } r1 == 0 \ \text{THEN jump to line number 42} \\
\text{jgtzn } r1 & \ 42 \quad \text{IF } r1 > 0 \ \text{THEN jump to line number 42} \\
\text{jltzn } r1 & \ 42 \quad \text{IF } r1 < 0 \ \text{THEN jump to line number 42} \\
\text{jnezn } r1 & \ 42 \quad \text{IF } r1 != 0 \ \text{THEN jump to line number 42} \\
\end{align*}
\]

**Unconditional** jump

\[
\begin{align*}
\text{jumpn } 42 \quad \text{Jump to program line # 42}
\end{align*}
\]

**Indirect** jump

\[
\begin{align*}
\text{jump r1} \quad \text{Jump to the line# stored in r1}
\end{align*}
\]
What function does this program implement?

**Example #2:**

**Screen**

-6 (input)

**RAM**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>read r1</td>
</tr>
<tr>
<td>1</td>
<td>jgtzn r1 7</td>
</tr>
<tr>
<td>2</td>
<td>setn r2 -1</td>
</tr>
<tr>
<td>3</td>
<td>mul r1 r1 r2</td>
</tr>
<tr>
<td>4</td>
<td>nop</td>
</tr>
<tr>
<td>5</td>
<td>nop</td>
</tr>
<tr>
<td>6</td>
<td>nop</td>
</tr>
<tr>
<td>7</td>
<td>write r1</td>
</tr>
<tr>
<td>8</td>
<td>halt</td>
</tr>
</tbody>
</table>

(Input: 6)
Exercise

1. Write a Hmmm program to compute the following for \( x \) given as user input and output the result to the screen:

   a) If \( x<0 \)  \[3x - 4\]
   b) else if \( x>0 \)  \[x / 5\]
   c) else  \[x^2+10 / 5\]