CS 5 Today

News in Brief

Psychics predict that there was no CS 5 lecture yesterday. Definitive proof of paranormal phenomena!

(Claremont AP): A group of psychics has made an extraordinary set of predictions that, one-by-one, are being corroborated by scientists. “It is indeed true that we didn’t have CS 5 yesterday,” said one CS 5 professor. The psychics have also predicted that fall break will occur sometime within the next 3-10 days.

Sports: HMC CS Professor to coach 2020 U.S. Olympic chocolate-eating team
Weather: 63.79% chance of weather today

Page 42 will no longer be published

Farm animals displace penguins, invade CS 5 notes

We’re going to milk this for all it’s worth!
Machine Language Versus…

Central Processing Unit (CPU)

Program Counter: 00000000
Instruction Register: 00110010
Register 0: 00000000
Register 1: 00000010
Register 2: 00000001
Register 15: 00000011

Memory Location:
- Binary: 00110010
- Base 10: 26
- Binary: 00011010
- Base 10: 18
- Binary: 10011101
- Base 10: 157
- Binary: 00000000
- Base 10: 0
- Binary: 00000001
- Base 10: 1
- Binary: 00000010
- Base 10: 2
- Binary: 00000011
- Base 10: 3
- Binary: 00000100
- Base 10: 4
- Binary: 11111111
- Base 10: 255

16 bits wide in Hmmm

8 bit address
- 00110010
- 00011010
- 10011101

8 bit data in

Read
Write

Memory

16 bits wide in Hmmm

r0 is always 0!
...Assembly Language!

Central Processing Unit (CPU)

- Program Counter: 00000000
- Instruction Register: 00110010
- Register 0: 00000000
- Register 1: 00000010
- Register 3: 00000001
- Register 15: 00000011

Memory Location

<table>
<thead>
<tr>
<th>Binary</th>
<th>Base 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>0</td>
</tr>
<tr>
<td>00000001</td>
<td>1</td>
</tr>
<tr>
<td>00000010</td>
<td>2</td>
</tr>
<tr>
<td>00000011</td>
<td>3</td>
</tr>
<tr>
<td>00000100</td>
<td>4</td>
</tr>
<tr>
<td>11111111</td>
<td>255</td>
</tr>
</tbody>
</table>

- add r3 r0 r2
- add r1 r2 r2
- mul r1 r3 r1

8 bit address

8 bit data in

Read

Write

Memory
Hmmm Assembly Language

```
add r2 r2 r2  
sub r2 r1 r4  
mul r7 r6 r2  
div r1 r1 r1  
setn r1 42   
addn r1 -1   
read r10     
write r1     
```

- `add r2 r2 r2`  
  \[ \text{reg2} = \text{reg2} + \text{reg2} \]
  
  crazy, perhaps, but used ALL the time

- `sub r2 r1 r4`  
  \[ \text{reg2} = \text{reg1} - \text{reg4} \]
  
  which is why it is written this way in python!

- `mul r7 r6 r2`  
  \[ \text{reg7} = \text{reg6} \times \text{reg2} \]

- `div r1 r1 r1`  
  \[ \text{reg1} = \text{reg1} / \text{reg1} \]
  
  INTEGER division—no remainders

- `setn r1 42`  
  \[ \text{reg1} = 42 \]
  
  you can replace 42 with anything from -128 to 127

- `addn r1 -1`  
  \[ \text{reg1} = \text{reg1} - 1 \]
  
  a shortcut

- `read r10`  
  read from keyboard and write to screen

- `write r1`  
  Each instruction (and many more) gets implemented for a particular processor and particular machine…
**Unconditional jump**

```
jumpn 42
```

Replaces the PC (program counter) with 42. “Jump to program line number 42.”

**Conditional jumps**

- `jeqzn r1` # IF $r1 == 0$ THEN jump to line number #
- `jgtzn r1` # IF $r1 > 0$ THEN jump to the location in #
- `jltzn r1` # IF $r1 < 0$ THEN jump to the location in #
- `jnezn r1` # IF $r1 != 0$ THEN jump to the location in #

**Register jump**

```
jump r r1
```

Jump to the line # stored in reg1!

This IS making me jumpy!
Worksheet
Feeling Jumpy?

1. Write an assembly-language program that reads one integer, X, as keyboard input into register r1. Then the program should compute \( X^2 + 3X + 4 \), leaving the result in register r13, and write it out.

2. Write an assembly-language program that reads two integers r1 and r2 as keyboard input. Then, the program should compute \( r1^{r2} \) in register r13 and write it out. You may assume that \( r2 \geq 0 \).
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System instructions</strong></td>
<td></td>
<td></td>
</tr>
<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>
<tr>
<td><strong>Setting register data</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>setn rX N</td>
<td>Set register rX equal to the integer N (-128 to +127)</td>
<td></td>
</tr>
<tr>
<td>addn rX N</td>
<td>Add integer N (-128 to 127) to register rX</td>
<td></td>
</tr>
<tr>
<td>copy rX rY</td>
<td>Set rX = rY</td>
<td>mov</td>
</tr>
<tr>
<td><strong>Arithmetic</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add rX rY rZ</td>
<td>Set rX = rY + rZ</td>
<td></td>
</tr>
<tr>
<td>sub rX rY rZ</td>
<td>Set rX = rY - rZ</td>
<td></td>
</tr>
<tr>
<td>neg rX rY</td>
<td>Set rX = -rY</td>
<td></td>
</tr>
<tr>
<td>mul rX rY rZ</td>
<td>Set rX = rY * rZ</td>
<td></td>
</tr>
<tr>
<td>div rX rY rZ</td>
<td>Set rX = rY / rZ (integer division; no remainder)</td>
<td></td>
</tr>
<tr>
<td>mod rX rY rZ</td>
<td>Set rX = rY % rZ (returns the remainder of integer division)</td>
<td></td>
</tr>
<tr>
<td><strong>Jumps!</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jumpn N</td>
<td>Set program counter to address N</td>
<td></td>
</tr>
<tr>
<td>jmpmr rX</td>
<td>Set program counter to address in rX</td>
<td>jump</td>
</tr>
<tr>
<td>jeqzn rX N</td>
<td>If rX == 0, then jump to line N</td>
<td>jeqz</td>
</tr>
<tr>
<td>jnezn rX N</td>
<td>If rX != 0, then jump to line N</td>
<td>jnez</td>
</tr>
<tr>
<td>jgtzn rX N</td>
<td>If rX &gt; 0, then jump to line N</td>
<td>jgtz</td>
</tr>
<tr>
<td>jltzn rX N</td>
<td>If rX &lt; 0, then jump to line N</td>
<td>jltz</td>
</tr>
<tr>
<td>calln rX N</td>
<td>Copy the next address into rX and then jump to mem. addr. N</td>
<td>call</td>
</tr>
<tr>
<td><strong>Interacting with memory (RAM)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>loadn rX N</td>
<td>Load register rX with the contents of memory address N</td>
<td></td>
</tr>
<tr>
<td>storen rX N</td>
<td>Store contents of register rX into memory address N</td>
<td></td>
</tr>
<tr>
<td>loadr rX rY</td>
<td>Load register rX with data from the address location held in reg. rY</td>
<td>loadi, load</td>
</tr>
<tr>
<td>storerr rX rY</td>
<td>Store contents of register rX into memory address held in reg. rY</td>
<td>storei, store</td>
</tr>
</tbody>
</table>
Why Assembly Language?

It’s only the foolish that never climb Mt. Fuji—or that climb it again.

Who writes most of the assembly language used?
The Compiler

A program that translates from human-readable language into assembly language and machine language

```plaintext
x = 6
y = 7
z = x*y
print z
```

the code

```plaintext
setn r1 6
setn r2 7
mul r3 r1 r2
write r3
```

assembly or byte-code

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

executable machine code

```plaintext
0000 0001 0010 0001
0110 0010 0010 0001
0000 0010 0000 0010
0000 0000 0000 0000
```

interpreting byte code

```plaintext
0000 0001 ...
0010 0001 ...
0110 0010 ...
1000 0001 ...
```

machine code

```plaintext
interpreting byte code
```

```plaintext
0000 0001 ...
0010 0001 ...
0110 0010 ...
1000 0001 ...
```
Examples

Core 2 Duo

Each processor has its own endearing idiosyncrasies...

\[
x = 6 \\
y = 7 \\
z = x \times y
\]

print z

The code

.LFB2:
    pushq  %rbp
.LCFI:
    movq   %rsp, %rbp
.LCFI1:
    subq   $16, %rsp
.LCFI2:
    movl   $6, -12(%rbp)
    movl   $7, -8(%rbp)
    movl   -12(%rbp), %eax
    imull  -8(%rbp), %eax
    movl   %eax, -4(%rbp)
    movl   -4(%rbp), %esi
    movl   $.LC0, %edi
    movl   $0, %eax
    call   printf
    leave
    ret

.LC0:
    .ascii "z is \%d\12\0"
    .text
    .align 2
    .globl _main
    _main:
        mflr  r0
        stmw  r30, -8(r1)
        stw  r0, 8(r1)
        stwu r1, -96(r1)
        mr  r30, r1
        bcl  20, 31, "L000000000000001$pb"
"L000000000000001$pb":
        mflr  r31
        li   r0, 6
        stw  r0, 64(r30)
        li   r0, 7
        stw  r0, 60(r30)
        lwz  r2, 64(r30)
        lwz  r0, 60(r30)
        mulw  r0, r2, r0
        stw  r0, 56(r30)
        addis r2, r31, ha16(LC0-"L0000000000001$pb")
        la  r3, i16(LC0-"L0000000000001$pb")(r2)
        lwz  r4, 56(r30)
        bl  L_printf$LDBLStub$stub
        lwz  r1, 0(r1)
        lwz  r0, 8(r1)
        mtlr  r0
        lmw  r30, -8(r1)
        blr
storer Goes TO Memory

Hmmm CPU

r1

... 42

r15

... 42

indirect store

Hmmm RAM

0
read r1

1
blah

2
blah

3
setn r15 42

4
storer r1 r15

5
blah

...
loadr Comes FROM Memory

Hmmm CPU

r1

Hmmm RAM

\begin{array}{|c|}
\hline
0 \\
1 \\
2 \\
3 \\
4 \\
5 \\
6 \\
7 \\
8 \\
\hline
\end{array}

\begin{array}{|c|}
\hline
\text{Blah blah blah} \\
\hline
\end{array}

\begin{array}{|c|}
\hline
\text{loadr r1 r15} \\
\hline
\end{array}

r15

\begin{array}{|c|}
\hline
42 \\
\hline
\end{array}

\text{...}

\text{loadr r1 r15}

\text{storer r1 r15}

\text{indirect}

\text{load}

\text{...}

\text{42}

\text{43}

\text{...}
A function call in python:

```python
def main():
    r1 = input()
    result = factorial(r1)
    print(result)

def factorial(r1):
    # do work
    return result
```

Hmmm's `calln` operation:

- read r1
- `calln r14 4`
- write r13
- halt
- do stuff and
- answer in r13
- `jumpr r14`
Factorial: *Function Call*

**Hmmm CPU**
- r0: 0
- r1: Input value: x
- r13: Final result - return value - in progress
- r14: Location / line to return to

**Hmmm RAM**
- 0: read r1
- 1: calln r14 4
- 2: write r13
- 3: halt
- 4: setn r13 1
- 5: jeqzn r1 9
- 6: mul r13 r13 r1
- 7: addn r1 -1
- 8: jumpn 5
- 9: jump r14

Input value: x
Final result - return value - in progress
loop
return
Which Factorial Is It?

def fac1():
    r1 = input()
    r13 = 1
    while r1 != 0:
        r13 = r13 * r1
        r1 += -1
        print r13
    return

def fac2(r1):
    if r1 == 0:
        return 1
    else:
        return r1 * fac2(r1-1)
Stretch Break: Ancient Rome Facts!

• More Roman swords have been found at archaeological sites outside the former borders of the Roman empire than inside its borders!

• The Romans and the Chinese were vaguely aware of each other, and there is actually some evidence that the Romans sent a diplomatic mission to China around 166 AD!

• Letters of recommendation are the most common surviving document from the Greco-Roman world.

• Monte Testaccio (Pottery Mountain) is a 35 meter high hill in Rome formed from the remains of 53 million jugs which together held 1.6 billion gallons of olive oil.

Monte Testaccio dig ➔

← The hill today

These facts brought to you by Mia!
def main():
    r1 = input()
    r13 = emma(r1)
    r13 = r13 + r1
    print(r13)
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

def sarah(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13
def main():
    r1 = input()  # r1=3
    r13 = emma(r1)  # emma(3)
    r13 = r13 + r1
    print(r13)
    return

def emma(r1):
    r1 = r1 + 1  # r1=4
    r13 = sarah(r1)  # sarah(4)  # r13=47
    r13 = r13 + r1  # r13=??
    return r13

def sarah(r1):
    r1 = r1 + 42  # r1=4
    r13 = r1 + 1  # r13=46
    return r13
    # return(47)
Function Calls…

def main():
    r1 = input()  # r1=3
    r13 = emma(r1)  # emma(3)
    r13 = r13 + r1
    print(r13)
    return

def emma(r1):
    r1 = r1 + 1  # r1=4
    r13 = sarah(r1)  # sarah(4) r13=47
    r13 = r13 + r1  # r13=??
    return r13

def sarah(r1):
    r1 = r1 + 42  # r1=4
    r13 = r1 + 1  # r13=46
    return r13  # return(47)
def main():
    r1 = input()  # r1=3
    r13 = emma(r1)  # emma(3)  r13=51
    r13 = r13 + r1  # r13=??
    print(r13)
    return

def emma(r1):
    r1 = r1 + 1  # r1=4
    r13 = sarah(r1)  # sarah(4)  r13=47
    r13 = r13 + r1  # r13=51
    return r13  # return(51)

def sarah(r1):
    r1 = r1 + 42  # r1=4
    r13 = r1 + 1  # r13=46
    return r13  # return(47)
Function Calls...

def main():
    r1 = input()  # r1=3
    r13 = emma(r1)  # emma(3)  r13=51
    r13 = r13 + r1  # r13=54
    print(r13)  # 54
    return

def emma(r1):
    r1 = r1 + 1  # r1=4
    r13 = sarah(r1)  # sarah(4)  r13=47
    r13 = r13 + r1  # r13=51
    return r13  # return(51)

def sarah(r1):
    r1 = r1 + 42  # r1=46
    r13 = r1 + 1  # r13=47
    return r13  # return(47)
The Stack!

AFLAC, ShmAFLAC!
Be careful up there!

Insert ("push")

Remove ("pop")
Watch carefully…

What if I don’t give a hoot?!
def main():
    r1 = input()  
    r13 = emma(r1)
    r13 = r13 + r1
    print(r13)
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

def sarah(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13
def main():
    r1 = input()  # r1 = 3
    r13 = emma(r1)
    r13 = r13 + r1
    print(r13)
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

def sarah(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13
def main():
    r1 = input()
    r1 = int(r1)  # Assuming input() returns a string, convert to integer
    r13 = emma(r1)
    r13 = r13 + r1
    print(r13)
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

def sarah(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13
```python
def main():
    r1 = input()
    r13 = emma(r1)
    r13 = r13 + r1
    print(r13)
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

def sarah(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13
```

The stack in RAM!

Hmmm code up here!
def main():
    r1 = input()  # r1=3
    r13 = emma(r1)
    r13 = r13 + r1
    print(r13)
    return

def emma(r1):
    r1 = r1 + 1  # r1=4
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

def sarah(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13

We object to this!!!
Now Without Pigs and Geese!

It was better with pigs and geese!

def main():
    r1 = input()
    r13 = emma(r1)
    r13 = r13 + r1
    print(r13)
    return

def emma(r1):
    r1 = r1 + 1
    r13 = sarah(r1)
    r13 = r13 + r1
    return r13

def sarah(r1):
    r1 = r1 + 42
    r13 = r1 + 1
    return r13