

Assemblers

Kevin Walsh
CS 3410, Spring 2010
Computer Science
Cornell University

See: P&H Appendix B.1-2

...

T: ADDI r4, r0, -1

BEQ r3, r0, B

ADDI r4, r4, 1

LW r3, 0(r3)

J T

NOP

B: ...

...

JAL L

nop

nop

L: LW r5, 0(r31)

ADDI r5, r5, 1

SW r5, 0(r31)

...

C

compiler

```
int x = 10;  
x = 2 * x + 15;
```

MIPS
assembly

assembler

```
addi r5, r0, 10  
mulr r5, r5, 2  
addi r5, r5, 15
```

machine
code

```
0010000000000010100000000000001010  
000000000000001010010100001000000  
0010000010100101000000000000001111
```

CPU

Circuits

Gates

Transistors

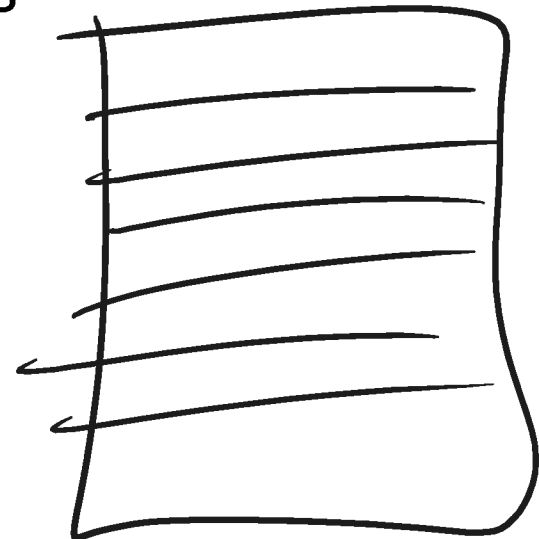
Silicon

[illegible]

Q: How to resolve labels into offsets and addresses?

A: Two-pass assembly

- 1st pass: lay out instructions and data, and build a *symbol table* (mapping labels to addresses) as you go
- 2nd pass: encode instructions and data in binary, using symbol table to resolve references



...
 JAL L
 nop
~~nop~~ Counter = 0
 L: LW r5, 0(r31)
 ADDI r5, r5, 1
 SW r5, 0(r31)
 ...

...
00100000000100000000000000000000100
00000000000000000000000000000000000
00000000000000000000000000000000000
10001111110010100000000000000000000
00001000101001010000000000000000001
00000000000000000000000000000000000
...

`.text 0x00400000 # code segment`

... *LA*

~~ori~~

~~ori r4, r0, counter~~

LW r5, 0(r4)

ADDI r5, r5, 1

SW r5, 0(r4)

...

`.data 0x10000000 # data segment`

counter:

.word 0

Lessons:

- Mixed data and instructions (von Neumann)
- ... but best kept in separate *segments*
- Specify layout and data using *assembler directives*
- Use *pseudo-instructions*

Pseudo-Instructions

NOP # do nothing

MOVE reg, reg # copy between regs

LI reg, imm # load immediate (up to 32 bits)

LA reg, label # load address (32 bits)

B label # unconditional branch

BLT reg, reg, label # branch less than

S L T
B N E

Assembler:

- assembly instructions
- + psuedo-instructions
- + data and layout directives
- = executable program

Slightly higher level than plain assembly

e.g: takes care of delay slots

(will reorder instructions or insert nops)

Q: Will I program in assembly?

A: I do...

- For kernel hacking, device drivers, GPU, etc.
- For performance (but compilers are getting better)
- For highly time critical sections
- For hardware without high level languages
- For new & advanced instructions: rdtsc, debug registers, performance counters, synchronization, ...

 x86 - Read Time Stamp Counter

