CS241 Lec 3

Graham Cooper

May 11th, 2015

Recall: Example 1: $\$3 \leftarrow \$5 + \$7$

Ex2: $\$3 \leftarrow 42 + 52$

lis $d - load immediate & skip

Location	Binary	Hex	Meaning
00000000	0000 0000 0000 0000 0010 1000 0001 0100	0x00002814	lis $5
00000004	0000 0000 0000 0000 0000 0000 0010 1010	0x0000002a	.word 42
00000008	0000 0000 0000 0000 0011 1000 0001 0100	0x00003814	lis $7
0000000c	0000 0000 0000 0000 0000 0000 0011 0100	0x00000034	.word 52

Then appaned code from Ex 1.

Assembly Language

- replace binary/hex encodings with easier to read shorthand
- less chance of error
- translation to binary can be automated (assembler)
- one line of assembly = 1 machine instruction (1 word)

EX2:

```
lis $5
.word 42
lis $7
.word 52
add $3, $5, 87
jr $31
```

.word is not an instruction, it is a directive that teslls the assembler that the next word in the file should be literally 42// jr $31 is return to loader

Ex3: Compute the absolute value of $1, store in $1, return

- some instructions modify PC
  - jumps, eg jr
  - branches
- beq
  - branch if 2 regs are equal
  - increment PC by the given # of words
  - can branch backwards
- also bne ”Branch not equal”
- ”set less than”

address	code	literal
00000000	slt $2, $1, $0	compare $1 <0
00000004	beq $2, $0, 1	if the above is false, skip over
00000008	sub $1, $0, $1	$1 ← -$1
0000000c	jr $31	exit program

Ex 4 (looping): Sum 1 ... 13 store in $3.

address	code	literal
0	lis $2	$2 ← 13
4	.word 13	
8	add $3, $0, $0	$3 ← 0
c	add $3, $2, $3	$3 += $2
10	lis $1	$1 ← 1
14	.word 1	
18	sub $2, $2, $1	–$2
1c	bne $2, $0, 5	if $2 ≠ $0 back to line c
20	jr $31	exit program

RAM

- lw ”load-word” from RAM to regs
  - lw $a, i($b)

  - $a ← MEM[$b + i]

- sw ”store-word” - from regs to RAM
  - sw $a, i($b)
  - MEM[$b + i] ← $a

Ex 5: $1 = address of an array, $2 = length of the array Place element 5 (0-based) into $3
Easy Way:

```
lw $3, 20($1)
jr $31
```

Hard Way:
Suppose $5 contains the index of the item we want to fetch

```
lis $4
.word 4
mult $5, $4
mflo $5
add $5, $1, $5
lw $3, 0($5)
jr $31
```

Mult Instruction Side Lesson

- mult $a, $b
- product of 2 32-bit #s is 64-bits (too big for a register)
- so two special registers, hi & lo to store the result of a mult
- $a, $b ≡ hi:lo ← $a × $b

Revisit looping example

```
ls $2
.word 13
add $3, $0, $0
add $3, $3, $2
lis $1 (move above add)
.word 1 (move above add)
sub $2, $2, $1
bne $2, $0, −5 (becomes −3)
jr $31
```

Moving instructions into/out of branches, we must update branch offsets, and can be tricky

Instead, the assembler allows labelled instructions
label : instr

eg: foo : add $1, $2, $3
Assembler associates the name ’foo’ with the address of the command add $1, $2, $3 in memory

```
ls $2
.word 13
add $3, $0, $0
top: add $3, $3, $2
lis $1
.word 1
sub $2, $2, $1
bne $2, $0, top
jr $31
```