MIPS Assembly Language Using AIK

EE685, Fall 2025

Hank Dietz

http://aggregate.org/hankd/

University of Kentucky

MIPS Registers ($ names)

$zero	0	constant 0
$at	1	reserved for assembler
$v0-$v1	2-3	value results
$a0-$a3	4-7	arguments (not on stack)
$t0-$t9	8-15,24-25	temporaries
$s0-$s7	16-23	save before use
$k0-$k1	26-27	reserved for OS kernel
$gp	28	global pointer (const)
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

MIPS ALU Instructions

- Either 3 reg operands or 2 regs and immediate 16-bit value (sign extended to 32 bits):

```
add  $rd,$rs,$rt      #rd=rs+rt
addi $rt,$rs,immed    #rt=rs+immed
```

- Suffix of **i** means immediate (**u** for unsigned)
- The usual operations: **add**, **sub**, **and**, **or**, **xor**
- Also has set-less-than, **slt**: rd=(rs<rt)

MIPS Load Immediate

- Can directly load a 16-bit immediate:

```
addi $rt,$0,imm   #rt=0+imm
```

- For 32-bit, generally use 2 instructions to load upper 16 bits then OR-in lower 16 bits:

```
lui $rt,imm       #rt=(imm<<16)
ori $rt,$rs,imm   #rt=rs|(imm&0xffff)
```

- MIPS assembler macro does it as **`li`** or **`la`**:

```
li $dest,const    #dest=const
```

MIPS Load & Store

- Can access a memory location given by a register plus a 16-bit Immediate offset:

```
lw $rt,off($rs)      #rt=memory[rs+off]
sw $rt,off($rs)      #memory[rs+off]=rt
```

- Byte and halfword using **b** and **h** instead of **w**

MIPS Jumps

- MIPS has a jump instruction, **`j`**:

```
j address                    #PC=address
```

- Call saves return address in **`$ra: jal addr`**
- Return is jump register using **`jr $ra`**
- Limited range (26 bits) for **`j`** or **`jal`**;
  can do full 32-bit target using jump register:

```
la $t0,address               #t0=address
jr $t0                       #PC=t0
```

MIPS Branches

- MIPS has only conditional branches:

```
beq $rs,$rt,lab  #if rs==rt, PC=lab
bne $rs,$rt,lab  #if rs!=rt, PC=lab
```

- The target is encoded as a 16-bit immediate:

```
immediate = (lab-(PC+4))>>2
```

- Branch over jump to target distant address

MIPS Comparisons

- Truth in C is “non-0,” so compare to **`$0`**
- Equality comparison can use **`xor`** or **`sub`**
- Inequality comparisons all use **`slt`**:

```
$t0=$t1<$t2     slt $t0,$t1,$t2
$t0=$t1>=$t2    ! $t0=$t1<$t2
$t0=$t1>$t2     slt $t0,$t2,$t1
$t0=$t1<=$t2    ! $t0=$t1>$t2
```

MIPS Assembler Notation

- One assembly directive or instruction per line
- # means to end of line is a comment…
  but AIK uses ;
- Labels look like they do in C, followed by a :
- Directives generally start with a .
- .word in AIK does bytes, not 32-bit words

```
.data              #the following is static data
.text              #the following is code
.word value        #initialize an 8-bit byte to value
```

MIPS AIK Assembler Notation

- MIPS AIK code to add x[0] = x[1] + x[2]

```
.data
x: .word   0,0,0,1          ; MIPS AIK word is 8 bits
   .word   0,0,0,6
   .word   0,0,0,7
.text
main:                       ; x[0] = x[1] + x[2]
   la $t0, x                ; t0 = &x
   lw $t1, 4($t0)           ; t1 = x[1]
   lw $t2, 8($t0)           ; t2 = x[2]
   addu $t3, $t1, $t2       ; t3 = t1 + t2
   sw $t3, 0($t0)           ; x[0] = t3
```

Code To Test An Instruction

- A trivial MIPS AIK program to test addu… assuming ori and beq are working

```
.text
main:
   ori $t0,1              ; t0 = 1
   ori $t1,2              ; t1 = 2
   ori $t2,4              ; t2 = 4
   ori $t3,6              ; t3 = 6
   addu $t0,$t1,$t2       ; test addu, $t0==6
   beq $t0,$t3,adduok     ; skip next if OK
.word 0,0,0,0             ; illegal instruction
adduok: ...
```