Tyler Thompson
Iorvenda Akem
Trevyn Kukin

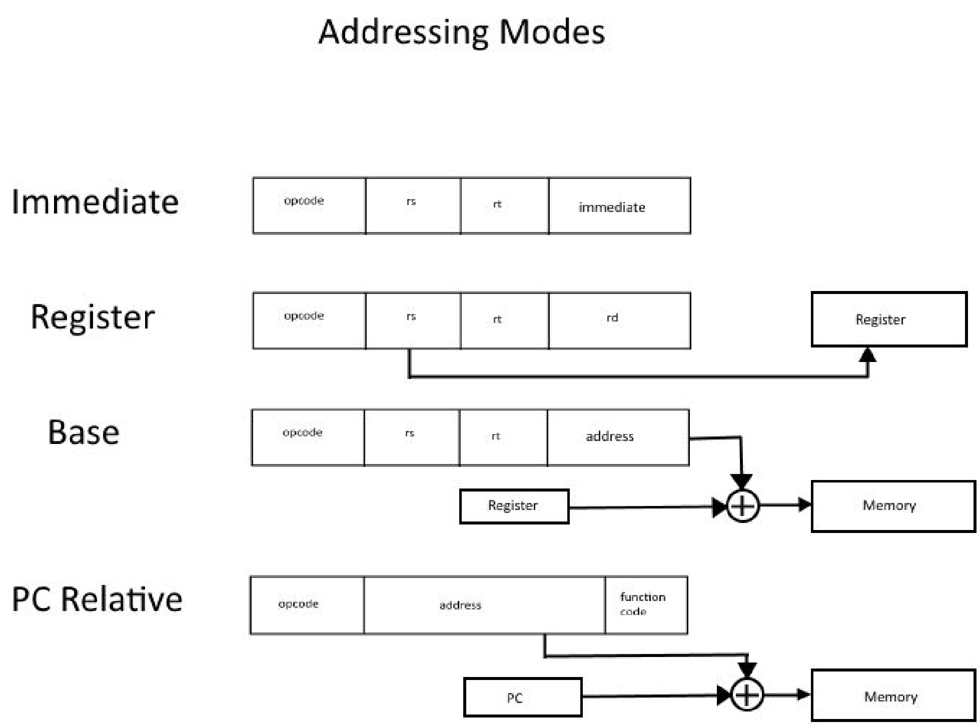
ECE 3570 Lab 1
10-bit Instruction Set Architecture
February 5th, 2018

Design Description

The reference sheet should explain how the design works, but there are a few things to note. The op-code is 3-bits long, so it can only allow for 8 instructions. The instruction set is extended by adding a 2-bit function code to the end of J type instructions. This extends the instruction set to allow for 10 instructions, but we are only using 9.

For R and I type instructions, the rs and rt codes are only 2-bits long, so $rs and $rt can only be assigned to registers 0 to 3. If you want to access a register above 3, you have to use and R type instruction. R type instructions allow for a 3-bit rd code.

I type instructions use 3-bits for immediate constants. Because it uses signed number, the max constant you can use is 3 and the min is -3.

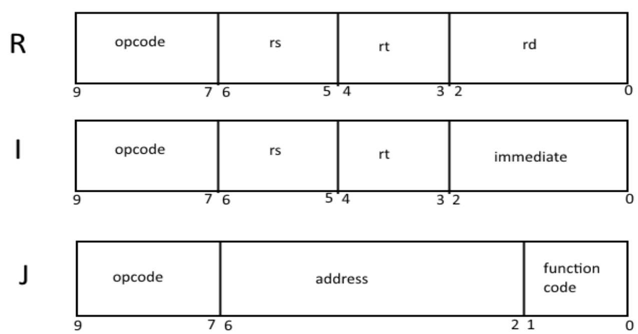


10-Bit ISA Reference Sheet

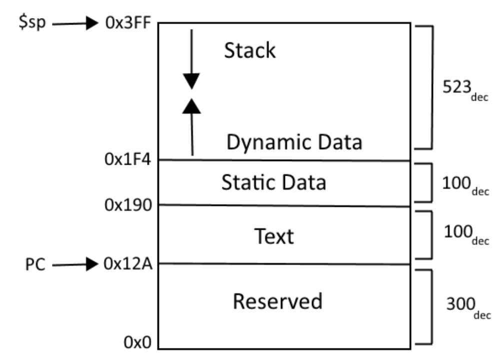
Instructions						
Name	**Description**	**Format**	**Operation**	**Syntax**	**OP Code (hex)**	**Function Code (hex)**
add	Add two values in registers together	R	$rs = $rt + $rd	add $rs, $rt, $rd	0x0	
addi	Add a 3-bit signed contant to a value in a register	I	$rs = $rt + imm	addi $rs, $rt, imm	0x4	
sw	Store a word in memory	I	M[$rs] = $rt	sw $rt, M($rs)	0x5	
lw	Load a word from memory	I	$rt = M[$rs]	lw $rt, M($rs)	0x6	
sll	Shift left logical	R	$rd = $rs << $rt	sll $rd, $rs, $rt	0x1	
slt	Store less than	R	$rd = ($rs < $rt)? 1:0	slt $rd, $rs, $rt	0x2	
beq	Branch equal	R	if($rs == $rt), PC=PC+4+$rd	beq $rd, $rs, $rt	0x3	
jal	Jump and link	J	$ra = PC+1, PC=address	jal Label	0x7	0x2
j	Jump	J	PC=PC + address	j Label	0x7	0x1
halt	Halt the machine	J	PC=HaltAddress	halt	0x7	0x3

Registers		
Name	**Number**	**Use**
$Zero	0	The constant 0
$t0	1	Temporary
$t1	2	Temporary
$s0	3	Saved Temporary
$sp	4	Stack Pointer
$a0	5	Argument
$v0	6	Function Return
$ra	7	Return Address

Instruction Formats



Memory Allocation



Notes

- $rs and $rt are only 2-bits wide, so they can only access registers 0 to 3. $rd is 3-bits wide and can access all 8.
- Jump instructions use 5-bit addresses, so they can only jump 32 instructions at a time.
- I type instructions use 3-bit signed constants, so addi can only add a max of 3 and a min of -3 to a value at a given time.
- Only J type instructions use function codes
- To load a value from registers 4 to 7, use: add $rs, $rt, $rd -> $rs = $rt + $rd
- To save a value into registers 4 to 7, use: sll $rd, $rs, $zero -> $rd = $rs << 0

Dynamic count: 23

```
Loop: addi $sp, $sp -8 // space for 5 elements
            sw $t0, 8($sp)
           sw $t1, 6($sp)
           sw $v0, 4($sp)
           sw $a0, 2($sp)
            sw $s0, 0($sp)
           sll $a0, $a0,1;
           sll $t0,$t0,1
            add $t0, $a0,$s0         // t0 holds a[n]
            addi $a0, $s0,1          // a0 holds n+1
           sll $t1,$t1,1
            add $t1,$a0,$s0          // t1 holds a[n+1]
            slt $v0, $t0, $t1        // if t0<t1 v0=1
           beq $t0,$t1      same
           beq $v0, $zero  swap    // branch to swap if its greater than
            j                    loop    // jumps back to loop if above its not true
swap: add $v0, $zero, $t0
            add $t0,$zero, $t1          // t0=a[n+1]
            add $t1,$zero, $v0      // t1=a[n]
            j                    loop
same:   add$t0,$zero,$t0
           add $t1,$zero,$t1
          addi, $a0,$a0,1               // $a0=n when n == the length this means the  last of them are the same
           beq  $a0, $t1         exit   // halts if the $a0==$t1  => means there are the last reg. of the array.
exit:     halt
```

Loading $a0 to $t0:

add $t0, $zero, $a0

Storing $t0 to $a0:

sll $a0, $t0, $zero

```
;ECE3570 10-Bit ISA Program 2
;Author: Tyler Thompson
;Date: 1/30/2018
;
;f = x*y - 4
;assuming x and y are stored in memory and $sp points to the base of
;the array they are stored in A[ x, y ]
;After the program runs, the result will be stored in A[2]
;The resulting array in stack will be A[ x, y, f ]
;
;Dynamic instruction count is 38


                    _start:
000 01 00 100                 add $t0, $zero, $sp     ;$t0 = $sp
110 01 11 000                 lw $s0, 0($t0)          ;$s0 = x
001 11 00 101                 sll $a0, $s0, $zero     ;$a0 = x
                    EvenOdd:
000 11 00 101                 add $s0, $zero, $a0     ;$s0 = test value
100 01 00 011                 addi $t0, $zero, 3
100 01 01 011                 addi $t0, $t0, 3
100 01 01 011                 addi $t0, $t0, 3        ;$t0 = 9
001 01 11 010                 sll $t1, $t0, $s0       ;$s0 << 9 store in $t1
100 01 00 111                 addi $t0, $zero, -3
100 01 01 111                 addi $t0, $t0, -3
100 01 01 111                 addi $t0, $t0, -3       ;$t0 = -9
001 10 01 110                 sll $v0, $t1, $t0       ;$v0 will be a 1->odd or a 0-
>even
000 01 00 110                 add $t0, $zero, $v0     ;$t0 = $v0, results of OddEven
011 01 00 111                 beq Even, $t0, $zero    ;if x is even, branch to Even
                     Odd:                             ;x is odd
100 01 11 101                 addi $t0, $s0, -1       ;calc x - 1 and store in $t0
100 10 00 101                 addi, $t1, $zero, -1    ;$t1 = -1
001 01 10 011                 sll $s0, $t0, $t1       ;$s0 = (x - 1) / 2
000 10 00 100                 add $t1, $zero, $sp     ;$t1 = $sp
110 10 01 001                 lw $t0, 1($t1)          ;$t0 = y
001 01 11 010                 sll $t1, $t0, $s0       ;$t1 = y << ((x - 1) / 2)
000 01 00 100                 add $t0, $zero, $sp     ;$t0 = $sp
110 01 11 001                 lw $s0, 1($t0)          ;$s0 = y
000 01 10 011                 add $t0, $t1, $s0       ;$t0 = x * y
001 10 00 101                 sll $a0, $t1, $zero     ;$a0 = x * y
111 01000 01                  j SubFour
                     Even:                            ;$s0 still = x, calc x / 2
100 10 00 101                 addi, $t1, $zero, -1    ;$t1 = -1
001 11 10 001                 sll $t0, $s0, $t1       ;$t0 = x / 2
000 11 01 000                 add $s0, $t0, $zero     ;$s0 = x / 2
000 01 00 100                 add $t0, $zero, $sp     ;$t0 = $sp
110 01 10 001                 lw $t1, 1($t0)          ;$t1 = y, load y
001 10 11 001                 sll $t0, $t1, $s0       ;$t1 = y << (x / 2)
001 01 00 101                 sll $a0, $t0, $zero     ;$a0 = x * y
                     SubFour:
100 01 00 111                 addi $t0, $zero, -3
100 01 01 101                 addi $t0, $t0, -1       ;$t0 = -4
000 11 01 101                 add $s0, $t0, $a0       ;$t1 = (x * y) -4
000 01 00 100                 add $t0, $zero, $sp     ;$t0 = $sp
101 01 11 010                 sw $s0, 2($t0)          ;A[2] = $s0
111 00000 11                  halt
```

```
; ECE3570 10-bit ISA Program 3
; Author: Trevyn Kukin
;
; Write a program that copies a null terminated string from one array in memory to
 another and returns number 10 in one of the registers after the copy is finished.
; assuming x and y are stored in memory and $sp points to the base.
; the arrays are stored in A[ X, Y].
; after the program runs, the result will be stored in the return address.
;$v0 = x, $a0 = y, $s0 = null, $t0 = I, $t1 = same
; dynamic instruction count is 110.


_start:
        Add $t0, $zero, $zero           ;I = 0
        Add $t1, $zero, $zero           ;same = 0
        Add $ra, $zero, $sp             ;$ra = $sp
        Add $ra, $zero, $zero           ;$ra = 0
        Addi $ra, $ra, 3                ;$ra = 3
        Addi $ra, $ra, 3                ;$ra = 6
        Addi $ra, $ra, 3                ;$ra = 9
        Addi $ra, $ra, 1                ;$ra = 10

L1:
        Add $t0, $zero, $sp             ;$t0 = $sp
        Lw $s0, 0($t0)                  ;$s0 = x
        Sll $v0, $s0, $zero             ;$v0 = x
        Add $t1, $zero, $sp             ;$t1 = $sp
        Lw $s0, 1($t1)                  ;$s0 = y
        Sll $a0, $s0, $zero             ;$a0 = y
        Beq $t0, $ra, exit              ;exit loop if I = 10
        Beq $v0, $s0, endstr            ;if x[I] = null branch to endstr
        Addi $t0, $t0, 1                ;I = I + 1
        J L1                            ;jump to L1
endstr:
        Addi $t1, $zero, 1              ;set same to 1
        Jr $Ra                          ;return 10 in the return address
Exit:
        Halt
```