


Addressing Modes

- dealing with small constants
- we have used small constants in add and subtract instructions
- UNfortunately, MIPS add and subtract require 3 registers
- so we really should have done a


```
lw    $t0, address of cell containing 4($zero)
add   $sp, $sp, $t0
```
- immediate addressing stores the constant in the instruction


```
addi  $sp, $sp, 4
```


Fall, 2008 Charlie Shub - CS216 - Addressing Page 1



Immediate addressing

- uses I format of instruction encoding
- I is for immediate
- But I format is also used for load, store, and conditional branch
- load has destination reg, base reg, offset
- store has source reg, base reg, offset
- beq has compare reg, compare reg, offset
- addi has destination reg, source reg, value


Fall, 2008 Charlie Shub - CS216 - Addressing Page 2



How about 32 bit values

- use two instructions
- load the top half
- add the bottom half
- lui (load upper immediate)
 - lui <destination> <value>
 - puts value in top 16 bits
 - zeros bottom 16 bits
- addi <destination> <source> <value>
 - doesn't always work, Why????
 - What are fixes?????


Fall, 2008 Charlie Shub - CS216 - Addressing Page 3



Right way to do 32 bit values

- use two instructions
- load the top half
- or in bottom half
- lui (load upper immediate)
 - lui <destination> <value>
 - puts value in top 16 bits
 - zeros bottom 16 bits
- ori <destination> <source> <value>


Fall, 2008 Charlie Shub - CS216 - Addressing Page 4



Using pseudo instructions

- can not say `addi $t0, $t0, 70000`
- so instead say (pseudoinst) `add $t0, $t0, 70000`
- can use hexadecimal or `$t0, $t0, 0xFF800000`
- can declare hex `.word 0xFF800000`

Fall, 2008 Charlie Shub - CS216 - Addressing Page 5



Far Branches

- a bne can only go up to 32,767 bytes away
- what if the distance is further
 - you probably have an UGLY program
 - you can workaround by changing


```
bne $s0, $s1, Far
```
 - to


```
beq $s0, $s1, Around
j Far # can get 256 Mb away
```
- Around:
 - to go further than 256MB, load a register with the 32 bit address and do a "jr"

Fall, 2008 Charlie Shub - CS216 - Addressing Page 6

UCCS Processor Addressing modes

- Register
 - operand value is found in the register
 - used for variables and temps
- Base/Displacement
 - add offset/displacement in instruction to address value in base register to get memory address where the operand is located
 - used for memory storage of variables and arrays
- Immediate
 - operand value is found in the instruction
 - used for small constants
- PC Relative
 - operand value is value in PC plus the 16-bit word-value(s) found in the instruction
 - used as branch destinations
- Pseudo-direct
 - operand value is top 4 bits of PC concatenated with 26-bit word-value found in the instruction, and 00, at the low end
 - Result: can only jump to the same 1/16 of memory.
 - Word boundary

Fall, 2008 Charlie Shub - CS216 - Addressing Page 7

UCCS Memory Addressing Mode

- Base/Displacement
 - the operand is in memory
 - the ADDRESS of the operand is found by adding the 16 bits found in the instruction to the thirty two bits found in the register
 - normally the register contains an address and the instruction contains a small integer known as a displacement or offset
 - historically, the register could contain an offset or subscript and the instruction could contain the base address, and this mode was called indexed mode

Fall, 2008 Charlie Shub - CS216 - Addressing Page 8

UCCS Picture of addressing Modes (figure 2.24 page152 [3rd ed 101])

Fall, 2008 Charlie Shub - CS216 - Addressing Page 9

UCCS Reverse Engineering

- Decoding machine language is described on pages 151-154 (100-104 in 3rd edition)
- this will not be discussed in class
 - Given 32 bits
 - 1st 6 bits identifies instruction and format
 - Then decode the rest of the bits per format
 - 5, 5, 16 (immediate instructions)
 - 5, 5, 5, 5, 6 (register instructions)
 - 26 (pseudo direct instructions)
- you will be responsible for the concepts on the tests

Fall, 2008 Charlie Shub - CS216 - Addressing Page 10

UCCS Pointers and arrays

```

cleara(int array[], int size)
{
    int i;
    for (i=0; i<size;i++) array[i]=0;
}
clearp(int *ray, int size)
{
    int *p;
    for (p=&ray[0]; p < &ray[size]; p++) *p = 0;
}
    
```

Fall, 2008 Charlie Shub - CS216 - Addressing Page 11


UCCS Array

- A0 and A1 have array and size
- i is stored in T0

```

move    $t0, $zero    # initialize i to 0
L: add  $t1,$t0,$t0    # you know what this is
add    $t1,$t1,$t1    # T1 has 4*i
add    $t2,$t1,$a0    # T2 has address of A[i]
sw     $zero, 0($t2)  # nuke it!!!!
addi   $t0, $t0, 1    # i++
slt    $t3, $t0, $a1  # T3 = I < size
bne    $t3, $zero,L   # and branch
jr     $ra            # and return
    
```

Fall, 2008 Charlie Shub - CS216 - Addressing Page 12




Pointer

- A0 and A1 have address and size
- start and end addresses are in T0 and T1

```

move $t0, $a0 # initialize pointer
add $t1, $a1, $a1 # compute end pointer as
add $t1, $t1, $t1 # 4*size + start pointer
add $t1, $t1, $a0
L: sw $zero, 0($t0) # nuke this one
addi $t0, $t0, 4 # bump the pointer
slt $t3, $t0, $t1 # is it less than the end
bne $t3, $zero, L # and branch
jr $ra # return
    
```


Fall, 2008 Charlie Shub - CS216 - Addressing Page 13



Comparison

- With pointer we can move the multiply by 4 code out of the loop because we only need to calculate the test value (address past the end of the array) one time
- with index, we must convert index to offset each time (or we could introduce a second variable and increment it by 4


Fall, 2008 Charlie Shub - CS216 - Addressing Page 14



Other Architectures

- Design alternative:
 - provide more powerful operations
 - goal is to reduce number of instructions executed
 - danger is a slower cycle time and/or a higher CPI
- Sometimes referred to as "RISC vs. CISC"
 - virtually all new instruction sets since 1982 have been RISC
 - VAX: minimize code size, make assembly language easy
instructions from 1 to 54 bytes long!
- Text looks at PowerPC and 80x86


Fall, 2008 Charlie Shub - CS216 - Addressing Page 15



Power PC

- Indexed addressing
 - example: `lw $t1, $a0+$s3 $t1=Memory[$a0+$s3]`
 - What do we have to do in MIPS?
- Update addressing
 - update a register as part of load (for marching through arrays)
 - example: `lwu $t0, 4($s3)`
`#implements $t0=Memory[$s3+4]; $s3=$s3+4`
 - What do we have to do in MIPS?
- Others:
 - load multiple/store multiple
 - a special counter register "bc Loop"
decrement counter, if not 0 goto loop

Fall, 2008 Charlie Shub - CS216 - Addressing Page 16



80x86


- 1978: The Intel 8086 is announced (16 bit architecture)
- 1980: The 8087 floating point coprocessor is added
- 1982: The 80286 increases address space to 24 bits, +instructions
- 1985: The 80386 extends to 32 bits, new addressing modes
- 1989-1995: The 80486, Pentium, Pentium Pro add a few instructions (mostly designed for higher performance)
- 1997: MMX is added

"This history illustrates the impact of the "golden handcuffs" of compatibility

"adding new features as someone might add clothing to a packed bag"

"an architecture that is difficult to explain and impossible to love"

Fall, 2008 Charlie Shub - CS216 - Addressing Page 17




What is Dominant


- Complexity:
 - Instructions from 1 to 17 bytes long
 - one operand must act as both a source and destination
 - one operand can come from memory
 - complex addressing modes
e.g., "base or scaled index with 8 or 32 bit displacement"
- Saving grace:
 - the most frequently used instructions are not too difficult to build
 - compilers avoid the portions of the architecture that are slow

"what the 80x86 lacks in style is made up in quantity, making it beautiful from the right perspective"

Fall, 2008 Charlie Shub - CS216 - Addressing Page 18




Fallacies and Pitfalls




- Instruction complexity is only one variable
 - lower instruction count vs. higher CPI / lower clock rate
- Design Principles:
 - simplicity favors regularity
 - smaller is faster
 - good design demands compromise
 - make the common case fast
- Instruction set architecture
 - a very important abstraction indeed!

Fall, 2008 Charlie Shub - CS216 - Addressing Page 19



Concluding Remarks



- Simplicity Favors Regularity
 - 1 instruction size
 - fields in same place
- Smaller is Faster
 - 32 registers is enough for most stuff
- Make the common case fast
 - PC relative addressing
 - Immediate addressing
- Good compromises
 - Fixed instruction length means
 - Can not have 32 bit addresses in instruction

Fall, 2008 Charlie Shub - CS216 - Addressing Page 20