Homework

• Reading
  – PAL, pp 127-152
• Machine Projects
  – MP2 due at start of Class 12
• Labs
  – Continue labs with your assigned section
Jumps and Flow of Control

• In assembly language, there are NO “if-else”, “for”, “do”, or “do … while” statements as in C
• Must use some combination of conditional and unconditional “jump” instructions for if-else branching or looping
• Jump instruction is similar to a C “go to”
• Jump instruction is similar to “call” instruction, but it doesn’t push a return address via %esp
Jumps and Flow of Control

• When the processor is fetching and executing instructions, it follows this cycle:
  – Fetches an instruction from memory using %eip
  – Executes the instruction
  – Increments %eip to point to the next instruction

• When a jump is executed, the last step of the fetch and execute cycle may be the loading of a different value into the %eip instead of the address for the next instruction in sequence
Jumps and Flow of Control

- Because there are no “structured programming” statements in assembly language, it may not be possible to use pseudo-code for the design.
- The design technique that best supports logic for an assembly language program is the flowchart.
- The flow chart has circles that represent labels and arrows that represent go-to’s.
Jumps and Flow of Control

• If-Else

```java
if (test)
    statement1;
else
    statement2;
```

One or more flags are set or reset

- Conditional Jump
  - **True**
    - `statement1`

  - **False**
    - `statement2`

- Unconditional Jump
Jumps and Flow of Control

- If-else in assembly code:

  cmpl $0, %eax  # test value of eax for zero
  jnz else

  ...               # statement1

  jmp end          # and jump over statement2

else:

  ...               # just a label

end:

  ...               # statement2

  ...               # statement2

end:               # next instruction after if-else
Jumps and Flow of Control

- **Iterative Loop**

  ```
  while (test) {
      body;
  }
  ```

  [Diagram showing iterative loop with decision points for conditional jumps and unconditional jumps, leading to the test condition being set or reset.]

  - Conditional Jump
    - True branch to body
    - False branch to Unconditional Jump

  - Unconditional Jump

  - One or more flags are set or reset
Jumps and Flow of Control

• While loop in assembly code:

```assembly
movl $3, %eax  # loop three times
while:
    # note – just a label
    cmpl $0, %eax  # test value of eax for zero
    jz end  # exit if counted down to zero
    ...  # body of loop here
    subl $1, %eax  # decrement eax
    jmp while  # loop
end:
    ...  # next instruction after loop
```
Unconditional Jumps

• “Unconditional jump” always loads %eip with a new value:
  – Hard coded address
    jmp 0x10ec  # hard coded address
    ...
  – Label for address
    jmp label   # address of a label
    ...
label:
An Infinite Loop

• The following is an infinite loop based on a single unconditional jump instruction:

```
movl $0, %eax
movl $2, %ecx
```

```
xz:
    addl %ecx, %eax
jmp xz
```
Conditional Jumps

• “Conditional jump” may or may not load %eip with a new value
• When your code performs instructions, specific flags in %eflag may get set (=1) or reset (=0)
• Depends on the definition of the instruction:

```
addb %bl, %al    # affects zero and carry flags
```

| %al | 1 0 1 1 0 0 0 1 |
| %bl | 1 0 0 1 0 0 1 0 |
| Carry Flag | 1 0 0 1 0 0 1 0 |
| Zero Flag | 0 1 0 0 0 0 1 1 |

1
Flags

- Flags are set by arithmetic or logical instructions:
  - Carry Flag – Set by a carry out / borrow in at MSB
  - Zero Flag – Set if entire byte, word, or long == 0
  - Sign Flag – Set if sign bit == 1
  - Parity Flag – Set if 8 LSB’s contain an even number of 1’s
  - Overflow Flag – Set by a carry into sign bit w/o a carry out
  - Auxiliary Carry Flag – Set by a carry / borrow in 4 LSBs
- These flags are individual bits in the %eflag register
- Specific flag settings control the behavior of specific conditional jump instructions
Conditional Jumps

• Operation of conditional jump:
  If (state of specific flags)
    Load a new value based on operand into %eip
  Else
    Let the %eip be incremented to next sequential instruction

• Examples:
  jz  label  # if zero flag is set
  js  label  # if sign flag is set
  jnz label  # if zero flag not set
  jns label  # if sign flag not set
  . . .
  label:
Conditional Jumps

- Be careful about the meaning of flag bits!
- C code:
  
  ```c
  if (al < bl) eax = 1; else eax = 0;               /* compute boolean value */
  ```
- Gas code (buggy):
  
  ```asm
  # assume values already in %al and %bl
  subb %bl, %al   # set/reset sign flag
  js             # jump if sign flag set
  movl $0, %eax  # %al is bigger or =
  jmp end        # don’t fall through
  sib: movl $1, %eax # %bl is bigger
  end: ret        # return value 0 or 1
  ```

- Bug is ignoring overflow flag!
Signed Comparisons

• Is it true?:
  A < B if and only if A – B is negative

• Not with fixed register sizes that can overflow!

Example test in signed character (1 byte) arithmetic:
Is 100 < -50?

No, but 100 - (-50) = -106 (Due to overflow!)

  100 01100100
- -50 + 00110010 (Add two’s compliment of -50)
  - 106 10010110 (Sets sign flag and sets overflow flag)

Note: Carry into sign bit without a carry out → Set overflow flag!
Signed Comparisons

• If overflow occurs, the sign flag value will be the opposite of what it should be!
• So we need our jump condition to be:
  – If overflow flag == 0, jump if sign flag == 1
  – If overflow flag == 1, jump if sign flag == 0
• Same as:
  – Jump if (sign flag XOR overflow flag) == 1
  – Hence, useful Intel instruction “jump less than”:
    jl label  # jump if (SF xor OV) is set
Signed Comparisons

- Proper interpretation of flag bits!
- C code:
  
  ```c
  if (al < bl) eax = 1; else eax = 0;               /* compute boolean value */
  ```

- Gas code (bug fixed for SIGNED data):
  
  ```
  # assume values already in %al and %bl
  subb %bl, %al  # set/reset sign flag
  jl sib         # jump less than
  movl $0, %eax  # %al is bigger or =
  jmp end        # don’t fall through
  sib: movl $1, %eax  # %bl is bigger
  end: ret        # return value 0 or 1
  ```
Signed Comparisons

• Compare Command
  – Sets the flags according to a subtraction
  – Does not save the result of the subtraction
  – Does not overwrite values in the registers being compared (just sets the flag bits)
Signed Comparisons

- Proper interpretation of flag bits!
- C code:
  
  ```c
  if (al < bl) eax = 1; else eax = 0;               /* compute boolean value */
  ```
- Gas code (using cmpb instead of subb):
  
  ```
  # assume values already in %al and %bl
  cmpb       %bl, %al       # set/reset flags
  jl         sib             # jump less than
  movl       $0, %eax       # %al is bigger or =
  jmp        end             # don’t fall through
  sib: movl     $1, %eax    # %bl is bigger
  end: ret     # return value 0 or 1
  ```
Conditional Jumps (Signed)

- **Jump**  
  - **jl**  less than  
  - **jle**  less than or equal  
  - **jg**  greater than  
  - **jge**  greater than or equal  
  - **je**  equal  
  - **jncc**  NOT of each of the above conditions
Unsigned Comparisons

• Is it true?:
  A < B if and only if A – B is “negative”

• Carry Flag will indicate underflow
  – Example test in unsigned character arithmetic:
  – Is 100 < 206? (206 = same bits as -50 was before)
  – Yes (because now the “sign bit” is $2^7$)

```
100 01100100
- 206 + 00110010   (Add two’s compliment of 206)
150 10010110   (Underflows = goes below zero)
```

Note: Underflow is a “Carry Error” → Set Carry flag!
Unsigned Comparisons

• Meaning of the carry flag is reversed
• A carry means a correct positive result after an unsigned subtraction, so carry flag = 0
• If underflow occurs, the carry flag = 1 will be indicator of an unsigned “negative” result!
• So we need our jump condition to be:
  – If carry == 1, jump
  – If carry == 0, don’t jump
• Hence, useful Intel instruction “jump below”:
  
  \texttt{jb \ label \ # \ jump \ if \ CF \ is \ set}
Unsigned Comparisons

• Proper interpretation of flag bits!
• C code:
  ```c
  if (al < bl) eax = 1; else eax = 0;                    /* compute boolean value */
  ```
• Gas code (bug fixed for UNSIGNED data):
  ```assembly
  # assume values already in %al and %bl
  cmpb %bl, %al    # set/reset carry flag
  jb sib          # jump below
  movl $0, %eax   # %al is bigger or =
  jmp end         # don’t fall through
  sib: movl $1, %eax # %bl is bigger
  end: ret        # return value 0 or 1
  ```
Conditional Jumps (Unsigned)

- Jump Condition
  - jb  below
  - jbe  below or equal
  - ja  above
  - jae  above or equal
  - je *  equal *
  - jncc  NOT of each of the above conditions
  - * Note: Same instruction as signed jump
loop Instruction

• Loop instruction = Decrement, Test, and Jump
• Instruction explanation:
  Decrement %ecx
  If %ecx != 0  
    Jump to label (Back to beginning of loop)
  Else
    Continue in sequence (Ends the loop)
• Example:
  movl $0x0a, %ecx  # loop 10 times
  label:
  (instructions in loop)
  loop label
  (next instruction after loop)
Scanning Pointer Problem

• Want to sum up the elements in an array of N elements

    .data
    iarray: .long 1, 4, 9, 16 # n = 4 in example

• The code might look like this:

    _sumarray: xorl %eax, %eax       # initial sum = 0
    movl $4, %ecx                    # initial loop count
    movl $iarray, %edx               # initial pointer value
    add1:    addl (%edx), %eax       # add in next element
             addl $4, %edx        # bump pointer
             loop add1           # test and loop
    ret
inc and dec Instructions

• Incrementing and decrementing by one
• Useful inside loops
• C code: (Inc/dec pointers by size of the data type!)
  
  i++; or i--;

• Gas incrementing and decrementing registers
  
  incl %eax    decl %eax

• Gas incrementing and decrementing memory
  
  incl index    decl index
  
  (Inc/dec pointers by one – not by size of the data type!)