Control Flow

CSE 132

Logistics

- Assignment 12 due date moved to April 27
  - Only one input set
  - In fixedPoint.ino, remove the "unsigned" type
  - Adding extra credit portion (not yet posted)
- Studio today
  - Flow control topics
  - Material is fair game for final exam
- Last lecture and studio will be review for final
  - Material will be cumulative
  - Tuesday, May 10, 10:30am-12:30pm, Lab Sci 300

Logistics (cont.)

- Late tickets
  - Even if you have consumed all of your late tickets, turn in (get checked out on) all the assignments
  - Partial credit ***is*** available, even if late
- Finishing assignments
  - All assignments must be checked out by **5:30pm on Wed., April 27**

Integer Multiplication

```
xxxxxxxxxxxxxxxx. * yyyyyyyyyyyyyyyy.

= zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz.

or                  wwwwwwwwwwwwwwww.
```

Q15 Multiplication

```
x.xxxxxxxxxxxxxxx * y.yyyyyyyyyyyyyyy

= zz.zzzzzzzzzzzzzzzzzzzzzzzzzzzzzz

or  w.wwwwwwwwwwwwwwww
```

Assembly Control Flow

- Unconditional Jump –

```
      jmp   [label]
```

e.g.,

```
      jmp   L1
            …
L1:   target instruction
            …
      ijmp          indirect, dest in Z
```

Conditional Control Flow

- In AVR, separate expression eval and cond branch inst.
- Compare –

```
cp    Rd, Rr
```

- Perform operation temp = Rd – Rr, throw away temp and set flags in SREG based on results of subtraction
- Flags can also be set as a result of normal arithmetic and/or logical operations

Conditional Jumps

```
br[cond]  [label]
```

e.g.,

```
brne      j_loop
```

- There are three classes of conditionals:
  - General (Simple)
  - Unsigned
  - Signed

General Conditionals

breq	zero (Z set)
brne	not zero (Z clear)
brcs	carry (C set)
brcc	no carry (C clear)

Signed Conditionals

brge	greater than or equal (Rd ≥ Rr)
brlt	less than (Rd < Rr)

Unsigned Conditionals

brsh	same or higher (Rd ≥ Rr)
brlo	lower than (Rd < Rr)

Control Flow in C

if … then …

```
if ([cond expr]) {
    [true body]
}
[main body]
```

e.g.,

```
if ( var1 > var2 ) {
    var1 = var1 + var2;
    var2 = 0;
}
…
```

if … then

if … then …

```
if ([cond expr]) {
    [true body]
}
[main body]
```

Assembly:

```
    cp  [cond exp opers]
    br[!cond] main_body
    [true body]
main_body:
    [main body]
```

if … then

```
if ( var1 > var2 ) {
   var1 = var1 + var2;
   var2 = 0;
}
…
```

```
        lds   r7, var1
        lds   r8, var2
        cp    r8, r7
        brle  main_body
        add   r7, r8
        sts   var1, r7
        sts   var2, r1
main_body:
        …
```

if … then … else

```
if ([cond expr]) {
    [true body]
}
else {
    [false body]
}
[main body]
```

```
        cp  [cond exp opers]
        br[!cond] false_body
        [true body]
        jmp main_body
false_body:
        [false body]
main_body:
        [main body]
```

if … then … else

```
if ( var1 == var2 ) {
   var1 = var1 + var2;
   var2 = 0;
}
else {
   var2 = var2 + var1;
   var1 = var2;
}
```

```
        lds   r7, (var1)
        lds   r8, (var2)
        cp    r8, r7
        brne  false_body
        add   r7, r8
        sts   (var1), r7
        sts   (var2), r1
        jmp   main_body
false_body:
        add   r8, r7
        sts   (var2), r8
        sts   (var1), r8
main_body:  …
```

Conditional if … then … else

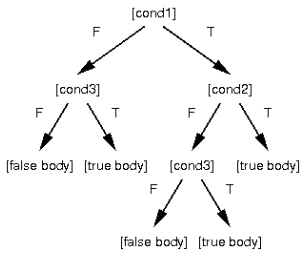
```
if (([cond1] && [cond2]) || [cond3]) {
      [true body]
}
else {
      [false body]
}
[main body]
```

- Note: evaluation order of compound expression is left to right, only conditions that need to be evaluated are evaluated

Conditional if … then … else

if (([cond1] && [cond2]) || [cond3])

Evaluation order for above compound expression:



if (([cond1] && [cond2]) || [cond3])

```
        cp          [cond1]
        br[!cond1]  check_cond3
        cp          [cond2]
        br[cond2]   true_body
check_cond3:
        cp          [cond3]
        br[cond3]   true_body
        [false body]
        jmp         main_body
true_body:
        [true body]
main_body:
        [main body]
```

for loop

```
for ([ind var] = [init val]; [cond expr]; [update ind var] ) {
        [loop body]
}
[main body]
```

e.g.,

```
for (i=0; i<24; i++) {
    mask = 1 << i;
    status_bit[i] = status & mask;
    status_bit[i] >>= i;
}
```

for loop

```
for ([ind var] = [init val]; [cond expr]; [update ind var] ) {
        [loop body]
}
[main body]
```

- Assembly

```
            ldi     [ind var], [init val]
for_loop:   cp      [cond expr]
            br[!cond]   loop_exit
            [loop body]
            [update ind var]
            jmp     for_loop
loop_exit:
            [main_body]
```

while loop

```
while ([cond expr]) {
        [loop body]
}
[main body]
```

- Assembly

```
        while_loop:
                cp          [cond expr oper]
                br[!cond]   exit_while
                [loop body]
                jmp         while_loop
        exit_while:
                [main body]
```