CS/ECE 252
2015 Nov 2nd

Assembly Programming and
Encoding Review
for Homework 6

Announcements

- Homework 6 due this Friday at 11AM
  - There were minor updates on PDF in red
  - Point value indicates which problems are and are not being graded

- Kai's Office Hours This Week (Nov 2nd – Nov 6th)
  - M 11:50 – 13:00, CS1240 or CS1308
  - TR 12:45 – 14:15, CS1308 (as usual)
  - R 19:00 – 23:00, CS1350 or CS1308
    - CS1350 is Enterprise Lab
  - F 08:00 – 10:45, CS1308

Outline

- Python to Assembly Translation
- Repeat with different control flow
- Repeat with an array
- Repeat with assembler directive to initialize array
- Repeat with function
- Repeat with caller-save
- Repeat with caller-save and reuse register
- Repeat with callee-save
- Assembly to Machine Translation
- In-class exercise: write divide function w/ 2 registers

Code Translation (Just Python1)

```
# in-class exercise
# translate from Python to AVR
i = 0
endIndex = 9
while(i < endIndex):
    print(i)
    i = i + 1
# end while loop
j = 45
```

Code Translation (Python1 to AVR1)

```
i = 0
endIndex = 9
while(i < endIndex):
    print(i)
    i = i + 1
# end while loop
j = 45
```

```
; r16 = i
; r17 = endIndex
; r18 = temp used for setting DDRD
; r19 = j
ldi r16, 0        ; r16 = i = 0
ldi r17, 9        ; r17 = endIndex = 9

begin_while_loop:
cp r16, r17       ; while(i < endIndex):
brsh end_while_loop
ldi r18, 255      ; print(i)
out 17, r18       ; ""
out 18, r16       ; ""
inc r16           ; i = i + 1
rjmp begin_while_loop

end_while_loop: ;# end while loop
ldi r19, 45       ; j = 45
```

Lesson: Can convert Python to AVR

Code Translation (Python1 to Python2)

```
i = 0
endIndex = 9
while(i < endIndex):
    print(i)
    i = i + 1
# end while loop
j = 45
```

```
i = 0
endIndex = 9
while(endIndex > i):
    print(i)
    i = i + 1
# end while loop
j = 45
```

Lesson: Can reverse operands to do the same thing

Code Translation (Python2 to AVR2)

```
i = 0
endIndex = 9
while(endIndex > i):
    print(i)
    i = i + 1
# end while loop
j = 45
```

```
; r16 = i
; r17 = endIndex
; r18 = temp used for setting DDRD
; r19 = j
ldi r16, 0      ; r16 = i = 0
ldi r17, 9      ; r17 = endIndex = 9

begin_while_loop:
cp r17, r16     ; while(endIndex > i):
brlo end_while_loop ; reverse everything?
breq end_while_loop
ldi r18, 255    ; print(i)
out 17, r18     ; ""
out 18, r16     ; ""
inc r16         ; i = i + 1
rjmp begin_while_loop

end_while_loop: ;# end while loop
ldi r19, 45     ; j = 45
```

Lesson: AVR can also reverse operands to do the same thing

Code Translation (Python1 to Python3)

```
i = 0
endIndex = 9
while(i < endIndex):
    print(i)
    i = i + 1
# end while loop
j = 45
```

```
i = 0
endIndex = 9
while(True):
    print(i)
    i = i + 1
    if(i >= endIndex):
        break
# won't work on edge case
# end while loop
j = 45
```

Lesson: Can translate while loops to do-while loops

Code Translation (Python3 to AVR3)

```
i = 0
endIndex = 9
while(True):
    print(i)
    i = i + 1
    if(i >= endIndex):
        break
# end while loop
j = 45
```

```
; r16 = i
; r17 = endIndex
; r18 = temp used for setting DDRD
; r19 = j
ldi r16, 0        ; r16 = i = 0
ldi r17, 9        ; r17 = endIndex = 9

begin_while_loop:
ldi r18, 255      ; print(i)
out 17, r18       ; ""
out 18, r16       ; ""
inc r16           ; i = i + 1
cp r16, r17       ; break if i >= endIndex
brsh end_while_loop
rjmp begin_while_loop

end_while_loop: ;# end while loop
ldi r19, 45       ; j = 45
```

Lesson: AVR can also do do-while loops by checking condition at end

Code Translation (AVR3 to AVR3-2)

```
; r16 = i
; r17 = endIndex
; r18 = temp used for setting DDRD
; r19 = j
ldi r16, 0       ; r16 = i = 0
ldi r17, 9       ; r17 = endIndex = 9

begin_while_loop:
ldi r18, 255     ; print(i)
out 17, r18      ; ""
out 18, r16      ; ""
inc r16          ; i = i + 1
cp r16, r17 ; break if i >= endIndex
brsh end_while_loop
rjmp begin_while_loop

end_while_loop: ;# end while loop
ldi r19, 45      ; j = 45
```

```
; r16 = i
; r17 = endIndex
; r18 = temp used for setting DDRD
; r19 = j
ldi r16, 0       ; r16 = i = 0
ldi r17, 9       ; r17 = endIndex = 9

begin_while_loop:
ldi r18, 255     ; print(i)
out 17, r18      ; ""
out 18, r16      ; ""
inc r16          ; i = i + 1
cp r17, r16      ; break if i >= endIndex
brlo end_while_loop
breq end_while_loop
rjmp begin_while_loop

end_while_loop: ;# end while loop
ldi r19, 45      ; j = 45
```

Lesson: AVR can check loop condition at end AND reverse operands to do the same thing

Code Translation (Python1 to Python4)

```
i = 0
endIndex = 9
while(i < endIndex):
    print(i)
    i = i + 1
# end while loop
j = 45
```

```
i = 0
endIndex = 9
array = [0,0,0,0,0,0,0,0,0]
while(i < endIndex):
    array[i] = i
    i = i + 1
i = 0
while(i < endIndex):
    print(array[i])
    i = i + 1
# end while loop
j = 45
```

Lesson: Can store and print an array

Code Translation (Python4 to AVR4)

```
i = 0
endIndex = 9
array = [0,0,0,0,0,0,0,0,0]
while(i < endIndex):
    array[i] = i
    i = i + 1
i = 0
while(i < endIndex):
    print(array[i])
    i = i + 1
# end while loop
j = 45
```

```
; r16 = i
; r17 = endIndex
; r18 = temp used for setting DDRD
; r19 = j
; r20 = value at array pointer
; r26 and r27 is used for memory pointer
ldi r16, 0              ; r16 = i = 0
ldi r17, 9              ; r17 = endIndex = 9
ldi r26, 0              ; set up memory pointer
ldi r27, 0

begin_while_loop1:
cp r16, r17             ; while(i < endIndex):
brsh end_while_loop1
mov r26, r16            ; update pointer to array
st X, r16               ; store i to array
inc r16                 ; i = i + 1
rjmp begin_while_loop1

end_while_loop1:        ;# end of 1st while loop
ldi r16, 0              ; r16 = i = 0

begin_while_loop2:
cp r16, r17             ; while(i < endIndex):
brsh end_while_loop2
mov r26, r16            ; update pointer to array
ld r20, X               ; read i from array
ldi r18, 255            ; print(i)
out 17, r18             ; ""
out 18, r20             ; ""
inc r16                 ; i = i + 1
rjmp begin_while_loop2

end_while_loop2:        ;# end while loop2
ldi r19, 45             ; j = 45
```

Lesson: AVR can also store and print an array

Code Translation (Python1 to Python5)

```
i = 0
endIndex = 9
while(i < endIndex):
    print(i)
    i = i + 1
# end while loop
j = 45
```

```
i = 0
endIndex = 9
array = [0,1,2,3,4,5,6,7,8]
while(i < endIndex):
    print(array[i])
    i = i + 1
# end while loop
j = 45
```

Lesson: Can initialize an array to print

Code Translation (Python1 to Python5)

```
i = 0
endIndex = 9
array = [0,1,2,3,4,5,6,7,8]
while(i < endIndex):
    print(array[i])
    i = i + 1
# end while loop
j = 45
```

```
; r16 = i
; r17 = endIndex
; r18 = temp used for setting DDRD
; r19 = j
; r20 = value at array pointer
.byte(array) 0,1,2,3,4,5,6,7,8
ldi r16, 0          ; r16 = i = 0
ldi r17, 9          ; r17 = endIndex = 9
ldi r26, lo8(array); set up array pointer
ldi r27, hi8(array)

begin_while_loop:
cp r16, r17         ; while(i < endIndex):
brsh end_while_loop
ldi r26, lo8(array); get array pointer
add r26, r16        ; go to ith index
ld r20, X           ; get value at array pointer
ldi r18, 255        ; print(i)
out 17, r18         ; ""
out 18, r20         ; ""
inc r16             ; i = i + 1
rjmp begin_while_loop

end_while_loop: ;# end while loop
ldi r19, 45         ; j = 45
```

Lesson: AVR can also initialize an array to print

Code Translation (Python1 to Python6)

```
i = 0
endIndex = 9
while(i < endIndex):
    print(i)
    i = i + 1
# end while loop
j = 45
```

```
def printI():
    print(i)
    return
i = 0
endIndex = 9
while(i < endIndex):
    printI()
    i = i + 1
# end while loop
j = 45
```

Lesson: Can wrap code in function

Code Translation (Python6 to AVR6)

```
def printI():
    print(i)
    return
i = 0
endIndex = 9
while(i < endIndex):
    printI()
    i = i + 1
# end while loop
j = 45
```

```
; r16 = i
; r17 = endIndex
; r18 = temp used for setting DDRD
; r19 = j
rjmp main

printI:
ldi r18, 255        ; print(i)
out 17, r18         ; ""
out 18, r16         ; ""
ret

main:
ldi r16, 0          ; r16 = i = 0
ldi r17, 9          ; r17 = endIndex = 9

begin_while_loop:
cp r16, r17         ; while(i < endIndex):
brsh end_while_loop
rcall printI        ; call function to print(i)
inc r16             ; i = i + 1
rjmp begin_while_loop

end_while_loop:     ;# end while loop
ldi r19, 45         ; j = 45
```

Lesson: Can translate function to AVR

Code Translation (AVR6 to caller-saved)

```
; r16 = i
; r17 = endIndex
; r18 = temp used for setting DDRD
; r19 = j
rjmp main

printI:
ldi r18, 255        ; print(i)
out 17, r18         ; ""
out 18, r16         ; ""
ret

main:
ldi r16, 0          ; r16 = i = 0
ldi r17, 9          ; r17 = endIndex = 9

begin_while_loop:
cp r16, r17         ; while(i < endIndex):
brsh end_while_loop
rcall printI        ; call function to print(i)
inc r16             ; i = i + 1
rjmp begin_while_loop

end_while_loop:     ;# end while loop
ldi r19, 45         ; j = 45
```

Lesson: Caller-saved

```
; r16 = i
; r17 = endIndex
; r18 = temp used for setting DDRD
; r19 = j
rjmp main

printI:
ldi r18, 255        ; print(i)
out 17, r18         ; ""
out 18, r16         ; ""
ret

main:
ldi r16, 0          ; r16 = i = 0
ldi r17, 9          ; r17 = endIndex = 9

begin_while_loop:
cp r16, r17         ; while(i < endIndex):
brsh end_while_loop
push r18            ; save r18
rcall printI        ; call function to print(i)
pop r18             ; restore r18
inc r16             ; i = i + 1
rjmp begin_while_loop

end_while_loop:     ;# end while loop
ldi r19, 45         ; j = 45
```

Code Translation (caller-saved w/ register reuse)

```
; r16 = i
; r17 = endIndex
; r18 = temp used for setting DDRD
; r19 = j
rjmp main

printI:
ldi r18, 255        ; print(i)
out 17, r18         ; ""
out 18, r16         ; ""
ret

main:
ldi r16, 0          ; r16 = i = 0
ldi r17, 9          ; r17 = endIndex = 9

begin_while_loop:
cp r16, r17         ; while(i < endIndex):
brsh end_while_loop
push r18            ; save r18
rcall printI        ; call function to print(i)
pop r18             ; restore r18
inc r16             ; i = i + 1
rjmp begin_while_loop

end_while_loop:     ;# end while loop
ldi r19, 45         ; j = 45
```

```
; r16 = i
; r17 = endIndex, reused to set up DDRD
; r18 = no longer needed
; r19 = j
rjmp main

printI:
ldi r17, 255        ; print(i)
out 17, r17         ; ""
out 18, r16         ; ""
ret

main:
ldi r16, 0          ; r16 = i = 0
ldi r17, 9          ; r17 = endIndex = 9

begin_while_loop:
cp r16, r17         ; while(i < endIndex):
brsh end_while_loop
push r17            ; save r18
rcall printI        ; call function to print(i)
pop r17             ; restore r18
inc r16             ; i = i + 1
rjmp begin_while_loop

end_while_loop:     ;# end while loop
ldi r19, 45         ; j = 45
```

Lesson: Can reuse registers with caller-saved

Code Translation (caller-saved to callee-saved)

```
; r16 = i
; r17 = endIndex
; r18 = temp used for setting DDRD
; r19 = j
rjmp main

printI:
ldi r18, 255        ; print(i)
out 17, r18         ; ""
out 18, r16         ; ""
ret

main:
ldi r16, 0          ; r16 = i = 0
ldi r17, 9          ; r17 = endIndex = 9

begin_while_loop:
cp r16, r17         ; while(i < endIndex):
brsh end_while_loop
push r18            ; save r18
rcall printI        ; call function to print(i)
pop r18             ; restore r18
inc r16             ; i = i + 1
rjmp begin_while_loop

end_while_loop:     ;# end while loop
ldi r19, 45         ; j = 45
```

```
; r16 = i
; r17 = endIndex
; r18 = temp used for setting DDRD
; r19 = j
rjmp main

printI:
push r18            ; save r18
ldi r18, 255        ; print(i)
out 17, r18         ; ""
out 18, r16         ; ""
pop r18             ; restore r18
ret

main:
ldi r16, 0          ; r16 = i = 0
ldi r17, 9          ; r17 = endIndex = 9

begin_while_loop:
cp r16, r17         ; while(i < endIndex):
brsh end_while_loop
rcall printI        ; call function to print(i)
inc r16             ; i = i + 1
rjmp begin_while_loop

end_while_loop:     ;# end while loop
ldi r19, 45         ; j = 45
```

Lesson: Callee-saved

Code Translation (callee-saved w/ register reuse)

```
; r16 = i
; r17 = endIndex
; r18 = temp used for setting DDRD
; r19 = j
rjmp main

printI:
push r18            ; save r18
ldi r18, 255        ; print(i)
out 17, r18         ; ""
out 18, r16         ; ""
pop r18             ; restore r18
ret

main:
ldi r16, 0          ; r16 = i = 0
ldi r17, 9          ; r17 = endIndex = 9

begin_while_loop:
cp r16, r17         ; while(i < endIndex):
brsh end_while_loop
rcall printI        ; call function to print(i)
inc r16             ; i = i + 1
rjmp begin_while_loop

end_while_loop:     ;# end while loop
ldi r19, 45         ; j = 45
```

```
; r16 = i
; r17 = endIndex, reused to set up DDRD
; r18 = no longer needed
; r19 = j
rjmp main

printI:
push r17            ; save r18
ldi r17, 255        ; print(i)
out 17, r17         ; ""
out 18, r16         ; ""
pop r17             ; restore r18
ret

main:
ldi r16, 0          ; r16 = i = 0
ldi r17, 9          ; r17 = endIndex = 9

begin_while_loop:
cp r16, r17         ; while(i < endIndex):
brsh end_while_loop
rcall printI        ; call function to print(i)
inc r16             ; i = i + 1
rjmp begin_while_loop

end_while_loop:     ;# end while loop
ldi r19, 45         ; j = 45
```

Lesson: Can reuse registers with callee-saved

Code Translation (Assembly to Machine) Part 1

```
; r16 = i
; r17 = endIndex
; r18 = temp used for setting DDRD
; r19 = j
ldi r16, 0        ; r16 = i = 0
ldi r17, 9        ; r17 = endIndex = 9

begin_while_loop:
cp r16, r17       ; while(i < endIndex):
brsh end_while_loop
ldi r18, 255      ; print(i)
out 17, r18       ; ""
out 18, r16       ; ""
inc r16           ; i = i + 1
rjmp begin_while_loop

end_while_loop: ;# end while loop
ldi r19, 45       ; j = 45
```

```
ldi r16, 0
ldi r17, 9
begin_while_loop:
cp r16, r17
brsh end_while_loop
ldi r18, 255
out 17, r18
out 18, r16
inc r16
rjmp begin_while_loop
end_while_loop:
ldi r19, 45
```

Step 1: remove comments and blank lines

Code Translation (Assembly to Machine) Part 2

```
ldi r16, 0
ldi r17, 9
begin_while_loop:
cp r16, r17
brsh end_while_loop
ldi r18, 255
out 17, r18
out 18, r16
inc r16
rjmp begin_while_loop
end_while_loop:
ldi r19, 45
```

```
ldi r16, 0
ldi r17, 9
; begin_while_loop:
cp r16, r17
brsh 5 ; end_while_loop
ldi r18, 255
out 17, r18
out 18, r16
inc r16
rjmp -7 ; begin_while_loop
; end_while_loop:
ldi r19, 45
```

Step 2: resolve all assembler directives

Code Translation (Assembly to Machine) Part 3

```
ldi r16, 0
ldi r17, 9
cp r16, r17
brsh 5
ldi r18, 255
out 17, r18
out 18, r16
inc r16
rjmp -7
ldi r19, 45
```

```
1110_0[7:4]_r16_0[3:0]
1110_9[7:4]_r17_9[3:0]
000101_r17[4]_r16_r17[3:0]
111101_5_000
ldi r18, 255
out 17, r18
out 18, r16
inc r16
rjmp -7
ldi r19, 45
```

Step 3: replace instruction with encoding

Code Translation (Assembly to Machine) Part 4

```
1110_0[7:4]_r16_0[3:0]
1110_9[7:4]_r17_9[3:0]
000101_r17[4]_r16_r17[3:0]
111101_5_000
ldi r18, 255
out 17, r18
out 18, r16
inc r16
rjmp -7
ldi r19, 45
```

```
1110_0000_0000_0000
1110_0000_0001_1001
000101_1_10000_0001
111101_0000101_000
ldi r18, 255
out 17, r18
out 18, r16
inc r16
rjmp -7
ldi r19, 45
```

Step 4: replace register and immediate encoding

Code Translation (Assembly to Machine)
Final Translation

```
; r16 = i
; r17 = endIndex
; r18 = temp used for setting DDRD
; r19 = j
ldi r16, 0       ; r16 = i = 0
ldi r17, 9       ; r17 = endIndex = 9

begin_while_loop:
cp r16, r17      ; while(i < endIndex):
brsh end_while_loop
ldi r18, 255     ; print(i)
out 17, r18      ; ""
out 18, r16      ; ""
inc r16          ; i = i + 1
rjmp begin_while_loop

end_while_loop: ;# end while loop
ldi r19, 45      ; j = 45
```

1110000000000000
1110000000011001
0001011100000001
1111010000101000
1110111100101111
1011101100100001
1011101100000010
1001010100000011
1100111111111001
1110001000111101

Lesson: Can convert AVR Assembly to Machine

In Class Exercise

- Write a function to integer divide by 5 using only 2 registers

```
; divide example from exam 2
function:
; y, dividend, is in r27
; result should be in r26
push r25
mov r25, r28
ldi r26, 0
begin_while:
cp r25, r27
brlo done_while
sub r25, r27
inc r26
rjmp begin_while
done_while:
pop r25
ret
```

```
# pseudo code for divide
quotient  = 0
while (dividend >= divisor):
    dividend = dividend – divisor
    quotient = quotient + 1
```

```
function:
; y, dividend, is in r27
; result should be in r26
push r27
ldi r26, 0 ; quotient  = 0

begin_while_loop:
; cpi r27, 5
subi r27, 5
brlo end_of_while_loop
; subi r27, 5
inc r26
rjmp begin_while_loop

end_of_while_loop:
pop r27
ret
```