Assembly Language and Exam 2 Review

CSE 132

1

Logistics

- Assignment 7 due next Monday (Mar 31)
  - No studio that day, available for help
- Exam 2 is next Wednesday (Apr 2)
  - Review today
  - Subject matter is modules 4 to 7
- Assignment 8 due Monday after exam (Apr 7)
  - That material will not be on exam 2
- Assignment 9 material is in today's lecture
  - Control flow in assembly language

2

General Form

```
label:  opcode  operands     comment
```

- Label is optional
- Opcode is the specific instruction (e.g., add)
- Operands specify data for operation
  - AVR is 2-operand machine, 1st operand is dest.
- Comments use different notations
  - Many assemblers (incl. AVR) ; comment
  - Or some other notation, e.g., # comment

3

Pseudo-operations

Pseudo-ops are commands to assembler

.text	means "text section", or instructions are next
.data	means "data section"
.byte	reserves data storage

```
var:   .byte   10
```

reserves one byte, initializes it to 10, and makes var a label that is address of byte

4

Example

```
byte rshift2(byte x) {
        return(x >> 2);
}

        .text           ;code segment follows
        .global rshift2 ;tell linker about rshift2
rshift2:
        lsr r24         ;do actual work
        lsr r24         ;result is in r24
        ldi r25, 0      ;return value in r25:r24
        ret             ;return
```

5

Assembly Control Flow

- Unconditional Jump –

```
     jmp   [label]
```

e.g.,

```
     jmp   L1
           …
  L1: target instruction
           …

     ijmp          indirect, dest in Z
```

6

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

7

Conditional Jumps

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

e.g.,

```
brne        j_loop
```

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

8

General Conditionals

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

9

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)

10

Control Flow in C

if … then …

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

e.g.,

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

11

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]
```

12

if … then

```
if ( var1 < var2 ) {                lds   r7, var1
   var1 = var1 + var2;              lds   r8, var2
   var2 = 0;                        cp    r7, r8
}                                   brge  main_body
…                                   add   r7, r8
                                    sts   var1, r7
                                    sts   var2, r1
                                 main_body:
                                    …
```

13

if … then (opposite test)

```
if ( var1 > var2 ) {                lds   r7, var1
   var1 = var1 + var2;              lds   r8, var2
   var2 = 0;                        cp    r8, r7
}                                   brge  main_body
…                                   add   r7, r8
                                    sts   var1, r7
But no brle instruction!            sts   var2, r1
                                 main_body:
                                    …
```

14

if … then … else

```
if ([cond expr]) {           cp  [cond exp opers]
     [true body]             br[!cond] false_body
}                            [true body]
else {                       jmp main_body
     [false body]          false_body:
}                            [false body]
[main body]                main_body:
                             [main body]
```

15

if … then … else

```
if ( var1 == var2 ) {               lds    r7, (var1)
   var1 = var1 + var2;              lds    r8, (var2)
   var2 = 0;                        cp     r8, r7
}                                   brne   false_body
else {                              add    r7, r8
   var2 = var2 + var1;              sts    (var1), r7
   var1 = var2;                     sts    (var2), r1
}                                   jmp    main_body
                               false_body:
                                    add    r8, r7
                                    sts    (var2), r8
                                    sts    (var1), r8
                               main_body:  …
```

16

Conditional if … then … else

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

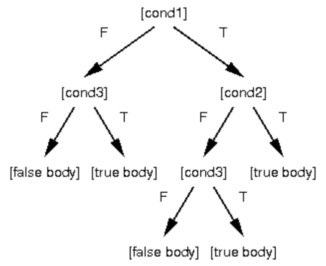
- Note: evaluation order of compound expression is left to right, only conditions that need to be evaluated are evaluated

17

Conditional if … then … else

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

Evaluation order for above compound expression:



18

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]
```

19

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;
}
```

20

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]
```

21

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]
```

22

Exam 2 Review

Exam is next week!

31

Logistics and Style

- Date and Time
  - Apr 2, 1pm to 2:20pm, starting right at 1!
  - Hillman 60 (regular lecture hall, right here)
  - Fill out top sheet when you arrive, but don't start yet
- Questions
  - Question 1 will be a collection of short answer things (e.g., true/false, multiple choice)
  - Questions 2 through N will be longer (going more in depth on a particular subject)
- One-page "crib sheet" allowed
  - 8.5" x 11" sheet, front and back, whatever you want to include (content-wise)

32

Help in Preparation

- Take practice exam
  - For extra credit
  - 100% of questions are from old exams
- Review B quizzes
  - They are designed to be in the same style as the exam
- Attend TA hours
- Ask questions on Piazza

33

Coverage

- Modules 4, 5, 6, and 7 are all fair game
  - Prep material for modules 4 to 7
  - Studios 4 to 7
  - Assignments 4 to 7
- Material from earlier that is needed to do modules 4, 5, 6, and 7 is still fair game
  - I.e., the test isn't designed to be comprehensive, but the material in the class *is* somewhat, so it can't be completely avoided

34

Communications

- Information representation
  - In Java vs. in Arduino vs. in comm. protocol
  - Integers, characters, strings
- Protocol design
  - Magic numbers, error recovery, debugging
  - Keys, what are they and what do they do for you?
  - Tradeoffs in protocol design choices

35

More Communications

- Stream concepts
  - Sequence of bytes
  - In-order delivery, no guaranteed delivery
- How to program on both platforms
  - Both sending and receiving (e.g., FSM receiver)
  - Both individual bytes and whole messages

36

Peripheral Devices

- Pushbuttons
  - Meaning, polarity
  - Physical construction
  - Debouncing
- Analog outputs
  - Pulse Width Modulation
- Time-based inputs
  - Ultrasonic distance measurement

37

Models of Computation

- Finite State Machines
  - Bubble diagrams
  - Simulation
  - Design for communications
- FSM Implementation
  - enums for state variable
  - Use of switch statement

38

Practicalities

- How to use development environment(s)
- Commonly used library functionality
  - Controlling pins (in and out)
  - Printing to attached PC
  - Timing
- Details of Arduino C language
  - Standard data types
  - Similarities and differences relative to Java
  - Bit-level and logical manipulation

39

Questions?

- Come early (just before 1pm) if you can, so we can start on time.

40