CprE 288 – Introduction to Embedded Systems
Course Review for Exam 3

Instructors:
Dr. Phillip Jones
Announcements

• Exam 3 scheduled time (75 minutes)
  – Morning class: Wed Dec. 16, 10:00am
  – Afternoon class: Thur Dec 17, 8:00am
• Exam 3 location: Our regular classroom
Topics in Assembly Programming

• Data Movement
  – Load immediate
  – Move between registers
  – Move between register and memory

• Logic & Arithmetic

• Control Flow
  – Test/compare register(s)
  – Choose the right branch

• Function call convention
  – Pass parameters and return values
  – Share registers between caller and callee
General-Purpose Registers

- **32 8-bit general purpose registers, R0-R31**
  - Used for instructions to execute operations on
  - Used for accessing SRAM
  - Used for passing function parameters and return value

- **What is an 8-bit register.**
  - Basically just 8 D-Flip connected together

- **Some special features to be aware of**
  - R16 – R31 can have immediate values loaded to them (e.g. LDI R17, 5), R0 – R15 cannot
  - R24 – R31 can be used with ADIW/SBIW, the other cannot
  - Register pairs R27:R26, R29:R28, R31:R30 can be used as 16-bit pointers (this pairs map to registers X, Y, Z)
Status Register (SREG)

**Z: Zero flag**, whether the result is zero

**C: Carry Flag**, whether a carry is generated

**S: Sign Bit**: *Actual* result is negative or not

**V: Two’s Complement Overflow Flag**

**N: Negative Flag**, whether the result is negative or not with signed operands

**H: Half Carry Flag**

**I: Global Interrupt Enable**

**T: Bit Copy Storage**
Data Movements

Register <= Immediate

\[ \text{LDI } Rd, \text{ imm} \quad ; \text{load immediate} \]

Register <= Register

\[ \text{MOV } Rd, \text{ Rr} \quad ; \text{Move register} \]
\[ \text{MOVW } Rd, \text{ Rr} \quad ; \text{Move word} \]

Note: 1 word is 2 bytes in AVR
Data Movements

Register <= Memory

- **LDS** \( Rd, \ addr \) ; load direct
- **LD** \( Rd, X/Y/Z \) ; load indirect
- **LD** \( Rd, X+/Y+/Z+ \) ; with post-incr.
- **LD** \( Rd, -X/-Y/-Z \) ; with pre-decr.
- **LDD** \( Rd, Y+q/Z+q \) ; with offset
Data Movements

Memory => Register

STS  \textit{addr, } \textit{Rr}  \quad; \text{store direct}

ST  \textit{X/Y/Z, } \textit{Rr}  \quad; \text{store indirect}

ST  \textit{X+/Y+/Z+, } \textit{Rr}  \quad; \text{with post-incr.}

ST  \textit{-X/-Y/-Z, } \textit{Rr}  \quad; \text{with pre-decr.}

STD  \textit{Y+q/Z+q, } \textit{Rr}  \quad; \text{with offset}
Data Movements

• Three **indirect address (pointer) registers**: X, Y, and Z
  
  X ↔ R27:R26
  Y ↔ R29:R28
  Z ↔ R31:R30

• **LDD/STD works for Y and Z only, not X**

• Other notes:
  
  – All load/store instructions take 2 cycles
  – LDS/STS is two-word (32-bits), others 1 word
  – q is from 0 to 63
Overview of arithmetic instructions

**Addition**: ADD, ADC, ADIW

**Subtraction**: SUB, SUBI, SBC, SBCI, SBIW

**Logic**: AND, ANDI, OR, ORI, EOR

**Compliments**: COM, NEG

**Register Bit Manipulation**: SBR, CBR

**Register Manipulation**: INC, DEC, TST, CLR, SER

**Multiplication**: MUL, MULS, MULSU

**Fractional Multiplication**: FMUL, FMULS, FMULSU

Give assemble op codes, give their binary encoding.

What is the binary encoding for “LDS r0, 0x0100”? 

Binary encoding format for LDS:

<table>
<thead>
<tr>
<th>1001</th>
<th>000d</th>
<th>dddd</th>
<th>0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>kkkk</td>
<td>kkkk</td>
<td>kkkk</td>
<td>kkkk</td>
</tr>
</tbody>
</table>

What is the binary encoding in hex?
char ch1 = 0x30;
char ch2 = 0x40;
int a = 0x1010;

ch1 = ch2;

a = ch1;
Example Questions

signed char ch1;
signed char ch2;
signed char flag;
int a;
int b;
signed char *pch;
int *pint;

*pch = ch1;

a = *pint;
pint = &b;

a = ch1 * ch2;

ch1 = ch1 & ch2;
extern unsigned int a, b;
extern int c;
extern unsigned char ch;

a = a - b;    // add 16-bit values

c = c + ch;   // add 8-bit to 16-bit
Conditional Branches

Commonly used branches

**BREQ**: EQ = Equal, signed or unsigned doesn’t matter

**BRNE**: N = Not Equal, signed or unsigned doesn’t matter

**BRLT**: L = Less Than, for signed type

**BRGE**: G = Greater than or E = Equal, for signed type

**BRLO**: L = Lower than, for unsigned type

**BRSH**: S = Same or H = Higher than, for unsigned type
If-Statement: Structure

Control and Data Flow Graph

Linear Code Layout

cond

F

T

if-body

test cond

br if cond=F

if-body
If-Else Statement: Structure

Control and Data Flow Graph

- **cond**
  - T: if-body
  - F: else-body

Linear Code Layout

1. **test cond**
2. **br if cond=F**
3. **If-body**
4. **jump**
5. **else-body**
Example Questions

extern int a, b, max;

if (a > b)
    max = a;
else
    max = b;

What if

extern unsigned int a, b, max
extern int a;
signed char flag;

if (a <= 20)
    flag = 0;
else
    flag = 1;
Caveats with Condition

Two types of if-conditions are trouble-free

- if \((a \geq b)\) branch if \(a < b\), use BRLT
- if \((a < b)\) branch if \(a \geq b\), use BRGE

What about

- if \((a > b)\) \iff if \((b < a)\)
- if \((a \leq b)\) \iff if \((b \geq a)\)
- if \((a > 10)\) \iff if \((a \geq 11)\)
- if \((a \leq 10)\) \iff if \((a < 11)\)
extern int a, b;
signed char flag;

if (flag)
a = b;
else
b = a;
AVR-GCC Call Convention: Parameters and Return Value

Function parameters
- R25:R24, R23:R22, ..., R9:R8
- All aligned to start in even-numbered register
  i.e. char will take two registers (use the even one)
- A long type uses two pairs
- Extra parameters go to stack

Function return values
- 8-bit in r24 (with r25 cleared to zero), or
- 16-bit in R25:R24, or
- 32-bit in R25-R22, or
- 64-bit in R25-R18
AVR-GCC Call Convention: Register Usage

How to share registers between caller and callee?

**Callee-save/Non-volatile**: R2-R17, R28-R29
   Caller may use them for free, callee must keep their old values

**Caller-save/Volatile**: R18-R27, R30-R31
   Callee may use them for free, caller must save their old values if needed

**Fixed registers**
   - R0: Temporary register used by gcc (no need to save)
   - R1: Should be zero
## AVR-GCC Call Convention

<table>
<thead>
<tr>
<th>Non-volatile: Callee save and restore if used</th>
<th>Volatile: Caller save and restore if used</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>R8 (P8)</td>
<td></td>
</tr>
<tr>
<td>R1 (Zero)</td>
<td>R9 (P8)</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>R10 (P7)</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>R11 (P7)</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>R12 (P6)</td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td>R13 (P6)</td>
<td></td>
</tr>
<tr>
<td>R6</td>
<td>R14 (P5)</td>
<td></td>
</tr>
<tr>
<td>R7</td>
<td>R15 (P5)</td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td>R16 (P4)</td>
<td></td>
</tr>
<tr>
<td>R9</td>
<td>R17 (P4)</td>
<td></td>
</tr>
<tr>
<td>R10</td>
<td>R18 (P3, V7)</td>
<td></td>
</tr>
<tr>
<td>R11</td>
<td>R19 (P3, V6)</td>
<td></td>
</tr>
<tr>
<td>R12</td>
<td>R20 (P2, V5)</td>
<td></td>
</tr>
<tr>
<td>R13</td>
<td>R21 (P2, V4)</td>
<td></td>
</tr>
<tr>
<td>R14</td>
<td>R22 (P1, V3)</td>
<td></td>
</tr>
<tr>
<td>R15</td>
<td>R23 (P1, V2)</td>
<td></td>
</tr>
</tbody>
</table>

P: Parameter  
V: Result
// Return the max of two integers
unsigned int
max(unsigned int a, unsigned int b);
Example Questions

• Make the the following function call
• Assume Y and Z registers have to be preserved

```
extern unsigned int a, b, c;
c = max(a, b);
```
Example Questions

PUSH r31 ; save Z-reg to stack
PUSH r30
LDS r24, a ; load 1st parameter
LDS r25, a+1
LDS r22, b ; load 2nd parameter
LDS r23, b+1
RCALL max ; call max
STS r24, c ; save return result
STC r25, c+1
POP r30 ; restore Z-reg from stack
POP r31

How about Y register?
DO-WHILE Loop

Control and Data Flow Graph

Linear Code Layout

Loop prologue (optional)

do-body

test cond

br if cond=T

Loop epilogue (optional)
WHILE Loop

Control and Data Flow Graph

Linear Code Layout

( optional prologue and epilogue not shown)
FOR Loop

Control and Data Flow Graph

Linear Code Layout

init-expr

for-body

incr-expr

cond

F

T

jump

for-body

Incr-expr

test cond

br if cond=T

(optional prologue and epilogue not shown)
Example Questions

// Copy the contents of array X into array Y.
// Both arrays have N elements.
void copyArray(int X[ ], int Y[ ], int N);
Example Questions

// Find out the maximum value of an array, return the value. The array has N elements
int maxOfArray(int X[], int N);