Lecture Topics

- Today: Assembly Language Basics
  (H&H 6.1-6.2)
- Next: Assembly Language Control
  (H&H 6.3)

Announcements

- Exam #1 results – sent via email
- Self-study Module #6
- Project #6 (due no later than 10/19)
- Project #7 (due no later than 10/26)
Project #6

- Major themes:
  - programming in C (using "gcc")
  - integer bitwise operations
  - floating point arithmetic
- Library module (function "mult")
- Driver module (function "main")

Project #6

- How to work with same 32 bits as both a floating point value and an integer value?
  - union
- How to unpack bit fields?
  - shifting and masking (with AND)
- How to pack bit fields back together?
  - shifting and masking (with OR)
union sp_union
{
    float frep;
    unsigned irep;
};

union sp_union num;
num.frep = 24.5;

printf( "Real: %f Hex: %08x\n", num.frep, num.irep );

Real: 24.500000 Hex: 41c40000

// Receive an object of type "float" and copy it to
// an object of type "union sp_union" so that it can
// be processed as an object of type "unsigned int"

void view_float( float value_to_view )
{
    union sp_union copy_of_value_to_view;

    copy_of_value_to_view.frep = value_to_view;

    printf( "%f (%08x)\n", copy_of_value_to_view.frep, copy_of_value_to_view.irep );
}
### Project #6 (unpacking)

```c
unsigned int value = 0x6789ABCD;
unsigned int field1;
unsigned int field2;

field1 = (value & 0x00FF0000) >> 16;
field2 = (value & 0x0000FF00) >> 8;

printf( "value: %08x\n", value );
value: 6789abcd
printf( "field1: %08x\n", field1 );
field1: 00000089
printf( "field2: %08x\n", field2 );
field2: 000000ab
```

### Project #6 (packing)

```c
unsigned int value1 = 0x1234;
unsigned int value2 = 0xABCD;
unsigned int result;

result = (value1 << 16) | value2;

printf( "value1: %08x\n", value1 );
value1: 00001234
printf( "value2: %08x\n", value2 );
value2: 0000abcd
printf( "result: %08x\n", result );
result: 1234abcd
```
ARM Machine Language

- Each ARM machine language instruction is 32 bits wide (4 bytes)
  - Operation code: bits 24:21
  - Destination register: bits 15:12
  - Source register #1: bits 19:16
  - Source register #2: bits 3:0

- What are the other bits used for?

Data processing instructions

Format:

- I bit
- S bit
- Rn
- Rd
- shifter_operand

<table>
<thead>
<tr>
<th>11</th>
<th>20</th>
<th>19</th>
<th>25</th>
<th>24</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>16</th>
<th>15</th>
<th>12</th>
<th>11</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- I bit: Distinguishes between the immediate and register forms of shifter_operand.
- S bit: Signifies that the instruction updates the condition codes.
- Rn: Specifies the first source operand register.
- Rd: Specifies the destination register.
- shifter_operand: Specifies the second source operand. See Addressing Mode 1 - Data-processing operands on page A5-2 for details of the shifter operands.
### Data processing instructions

**Opcodes:**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td><strong>and</strong></td>
<td>1000</td>
<td><strong>tst</strong></td>
</tr>
<tr>
<td>0001</td>
<td><strong>eor</strong></td>
<td>1001</td>
<td><strong>teq</strong></td>
</tr>
<tr>
<td>0010</td>
<td><strong>sub</strong></td>
<td>1010</td>
<td><strong>cmp</strong></td>
</tr>
<tr>
<td>0011</td>
<td><strong>rsb</strong></td>
<td>1011</td>
<td><strong>cmn</strong></td>
</tr>
<tr>
<td>0100</td>
<td><strong>add</strong></td>
<td>1100</td>
<td><strong>orr</strong></td>
</tr>
<tr>
<td>0101</td>
<td><strong>adc</strong></td>
<td>1101</td>
<td><strong>mov</strong></td>
</tr>
<tr>
<td>0110</td>
<td><strong>sbc</strong></td>
<td>1110</td>
<td><strong>bic</strong></td>
</tr>
<tr>
<td>0111</td>
<td><strong>rsc</strong></td>
<td>1111</td>
<td><strong>mvn</strong></td>
</tr>
</tbody>
</table>

- **S bit:** when the S bit is 1, NZCV condition code flags are updated by the instruction
- **I bit:** when I bit is 1, the second operand is an 8-bit immediate value (rather than a register)
- **cond:** will always be 1110 (for now)
Example: S bit

- \( r_{10} \leftarrow r_{6} + r_{7} \)
  
  \[
  \text{add } r_{10}, r_{6}, r_{7} \\
  e086a007
  \]

- \( r_{10} \leftarrow r_{6} + r_{7}, \text{update NZCV flags} \)
  
  \[
  \text{adds } r_{10}, r_{6}, r_{7} \\
  e096a007
  \]

Example: I bit

- \( r_{0} \leftarrow r_{6} + 1 \)
  
  \[
  \text{add } r_{0}, r_{6}, #1 \\
  e2860001
  \]

- \( r_{0} \leftarrow r_{6} + 0xff \)
  
  \[
  \text{add } r_{0}, r_{6}, #0xff \\
  e28600ff
  \]
Decoding machine language

- assembly:   \texttt{add r0, r6, \#0xff}
- machine language: \texttt{e28600ff}
- machine language in binary:

\begin{center}
\begin{tabular}{c|c|c|c|c}
 & & & & \\
& & & & \\
I bit & S bit & ADD & r6 & r0 & 0xff \\
& & & & \\
\end{tabular}
\end{center}

```
0b11100010100001100000000011111111
```


---

ARM Assembly Language

Express instructions in symbolic form, have software translate them into machine language instructions

Instructions: text file
Software: assembler

Assembler reads text file (".s" file), outputs to binary file (".o" file)
Example #12

Assembly language version of:

```
int A = 12;
int B = 27;
int C = -1;

int main()
{
    C = A + B;
}
```

```
.global main

.text

main:  ldr    r0, =A          @ get address of A
       ldr    r1, [r0]        @ load value of A
       ldr    r2, =B          @ get address of B
       ldr    r3, [r2]        @ load value of B
       add    r4, r1, r3      @ sum = A + B
       ldr    r5, =C          @ get address of C
       str    r4, [r5]        @ store sum into C

exit:   bx    lr         @ return from main
```
Example #13

Demonstrates bitwise instructions

Based on code segment in Harris and Harris textbook (page 304)

Course website:

~cse320/Examples/example13.pdf

Example #14

Demonstrates shift instructions

Based on code segment in Harris and Harris textbook (page 305)

Course website:

~cse320/Examples/example14.pdf