CS1021 After Reading Week

Mid-Semester Test

• NOW Thurs 8th Nov @ 9am in Goldsmith Hall (ALL students to attend at 9am)

Final 2 Labs

• Lab5 2-Nov-18, due 16-Nov-18 (2 weeks duration)
• Lab6 16-Nov-19, due 30-Nov-18 (2 weeks duration)

End of Semester Exam

• written 2 hour exam (week of 12-Dec-18)
• answer 3 out of 4 questions (not 2 out of 3 as in recent CS1021 exams)
• 40 mins per question
• questions similar to previous CS1021 exams (shorter, less parts)

Yet to cover

• reading and writing to memory, stacks and subroutines
ARM Memory System

- ARM system comprises CPU and memory
- Instructions and data stored in memory
- CPU can read (LOAD) data from memory into a register
- CPU can write (STORE) data from a register into memory
- Called a load / store architecture
- To operate on data in memory, the data must first be loaded into register(s), updated in the registers and then stored back to memory
Memory Revision

- memory comprises an array of memory locations

- each location stores a byte of data

- each location has a unique 32 bit address
  0x00000000 to 0xFFFFFFFF

- the address space, $2^{32}$ bytes (4GB), is the amount of memory that can be physically attached to the CPU

memory as an array of BYTES
Memory Revision...

- often easier to view memory as an array of WORDs (32 bits) rather than an array of BYTES

- as each WORD location is aligned on a 4 byte boundary, the low order 2 bits of each address is 0

- making a comparison with the previous slide, the byte of data stored at memory location 0 is the least significant byte of the WORD stored in location 0

- this way of storing a WORD is termed LITTLE ENDIAN - the least significant byte is stored at the lowest address (the other way is BIG ENDIAN)

- ARM CPUs can be configured to be LITTLE ENDIAN or BIG ENDIAN (term from Gulliver’s Travels)
NXP LPC2468 Memory Map

- address space NOT fully populated with memory devices
- 512K of flash memory at address 0x00000000 to 0x0007FFFF
- 64K RAM at address 0x40000000 to 0x4000FFFF
- flash memory
  - read ONLY (programmed electronically - “flashed”)
  - retains data when power removed
- RAM (random access memory)
  - read write
  - looses its data when power removed
- uVision projects are configured to simulate this memory map
- code placed in flash memory starting at address 0x00000000

NXP LPC2468 memory map (NOT to scale)
Load Instructions - LDR and LDRB

• memory address specified in a register

• load word

LDR R1, =0x40000000 ; R1 -> 0x40000000 (in RAM)
LDR R0, [R1] ; R0 = MWORD[0x40000000]

• load byte

LDR R1, =0x40000003 ; R1 -> 0x40000003 (in RAM)
LDRB R0, [R1] ; R0 = MBYTE[0x40000003]
LDR and LDRB

- **load word**
  - reads 4 bytes from memory address into a register
  - address must be even (LS address bit = 0)
  - normally used with an address aligned on a 4 byte boundary (address ends with ...00₂) BUT ...
  - if address end with ...10₂, it accesses memory as though the address ended with ...00₂ but swaps the high and low 16 bits

- **load byte**
  - reads byte from memory address and stores in LS byte of register
  - clears MS bytes of register
LDR and LDRB...

- load word

```
LDR  R1, =0x40000000  \( \text{RO} \) 0x04030201 
LDR  R0, [R1] 

LDR  R1, =0x40000003  \( \text{RO} \) 0x00000004 
LDRB R0, [R1] 
```

- load byte

- load word (address ends with ..10<sub>2</sub>)

```
LDR  R1, =0x40000002  \( \text{RO} \) 0x02010403 
LDR  R0, [R1] 
```

little endian
0x01 @ MBYTE[0x40000000]  
0x04 @ MBYTE[0x40000003]

high and low 16 bits swapped
Store Instructions – STR and STRB

- memory address specified in a register

- store word

  LDR  R1, =0x40000000  ; R1 -> 0x40000000 (in RAM)
  STR  R0, [R1]        ; MWORD[0x40000000] = R0

- store byte

  LDR  R1, =0x40000003  ; R1 -> 0x40000003 (in RAM)
  STRB R0, [R1]        ; MBYTE[0x40000003] = R0 (LS byte)
STR and STRB

- store word
  - writes ALL 4 bytes of register to memory address
  - address must be aligned on a 4 byte boundary (address ends with ...00₂)

- store byte
  - writes LS byte of register to memory address
Example

• a, b and c are 32-bit signed binary integers stored in memory locations 0x40000000, 0x40000004 and 0x40000008 respectively

• compute c = a + b

```
LDR   R1, =0x40000000   ; R1 -> a
LDR   R0, [R1]         ; R0 = a
LDR   R1, =0x40000004   ; R1 -> b
LDR   R1, [R1]         ; R1 = b
ADD   R0, R0, R1       ; R0 = a + b
LDR   R1, =0x40000008   ; R1 -> c
STR   R0, [R1]         ; c = a + b
```
ASCII strings

• American Standard Code for Information Interchange

• ASCII is a standard used to encode alphanumeric and other characters

• each character is stored as a single byte (8 bits)

• upper and lower case characters have different ASCII codes

• ASCII only uses 7 bits to encode the character, giving 128 possible characters

• MSB may be used as a parity bit
  
  • ODD or EVEN parity
  • parity bit set so that number of 1 bits in a character is either odd or even
  • used to detect transmit and receive errors

• originally used to transmit characters from a computer to a tele printer (terminal)
### ASCII Table (hex values)

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><em>O</em></td>
<td><em>1</em></td>
<td><em>2</em></td>
<td><em>3</em></td>
<td><em>4</em></td>
<td><em>5</em></td>
<td><em>6</em></td>
<td><em>7</em></td>
<td><em>8</em></td>
<td><em>9</em></td>
<td><em>A</em></td>
<td><em>B</em></td>
<td><em>C</em></td>
<td><em>D</em></td>
<td><em>E</em></td>
</tr>
<tr>
<td>0</td>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
<td>BS</td>
<td>HT</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
</tr>
<tr>
<td></td>
<td>0000</td>
<td>0001</td>
<td>0002</td>
<td>0003</td>
<td>0004</td>
<td>0005</td>
<td>0006</td>
<td>0007</td>
<td>0008</td>
<td>0009</td>
<td>000A</td>
<td>000B</td>
<td>000C</td>
<td>000D</td>
<td>000E</td>
</tr>
<tr>
<td>1</td>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
<td>RS</td>
</tr>
<tr>
<td></td>
<td>0010</td>
<td>0011</td>
<td>0012</td>
<td>0013</td>
<td>0014</td>
<td>0015</td>
<td>0016</td>
<td>0017</td>
<td>0018</td>
<td>0019</td>
<td>001A</td>
<td>001B</td>
<td>001C</td>
<td>001D</td>
<td>001E</td>
</tr>
<tr>
<td>2</td>
<td>SP</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>0020</td>
<td>0021</td>
<td>0022</td>
<td>0023</td>
<td>0024</td>
<td>0025</td>
<td>0026</td>
<td>0027</td>
<td>0028</td>
<td>0029</td>
<td>002A</td>
<td>002B</td>
<td>002C</td>
<td>002D</td>
<td>002E</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
</tr>
<tr>
<td></td>
<td>0030</td>
<td>0031</td>
<td>0032</td>
<td>0033</td>
<td>0034</td>
<td>0035</td>
<td>0036</td>
<td>0037</td>
<td>0038</td>
<td>0039</td>
<td>003A</td>
<td>003B</td>
<td>003C</td>
<td>003D</td>
<td>003E</td>
</tr>
<tr>
<td>4</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>0040</td>
<td>0041</td>
<td>0042</td>
<td>0043</td>
<td>0044</td>
<td>0045</td>
<td>0046</td>
<td>0047</td>
<td>0048</td>
<td>0049</td>
<td>004A</td>
<td>004B</td>
<td>004C</td>
<td>004D</td>
<td>004E</td>
</tr>
<tr>
<td>5</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>@</td>
</tr>
<tr>
<td></td>
<td>0050</td>
<td>0051</td>
<td>0052</td>
<td>0053</td>
<td>0054</td>
<td>0055</td>
<td>0056</td>
<td>0057</td>
<td>0058</td>
<td>0059</td>
<td>005A</td>
<td>005B</td>
<td>005C</td>
<td>005D</td>
<td>005E</td>
</tr>
<tr>
<td>6</td>
<td>`</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
</tr>
<tr>
<td></td>
<td>0060</td>
<td>0061</td>
<td>0062</td>
<td>0063</td>
<td>0064</td>
<td>0065</td>
<td>0066</td>
<td>0067</td>
<td>0068</td>
<td>0069</td>
<td>006A</td>
<td>006B</td>
<td>006C</td>
<td>006D</td>
<td>006E</td>
</tr>
<tr>
<td>7</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>0070</td>
<td>0071</td>
<td>0072</td>
<td>0073</td>
<td>0074</td>
<td>0075</td>
<td>0076</td>
<td>0077</td>
<td>0078</td>
<td>0079</td>
<td>007A</td>
<td>007B</td>
<td>007C</td>
<td>007D</td>
<td>007E</td>
</tr>
</tbody>
</table>

- ‘H’ = 0x48
- ‘e’ = 0x65
- ‘l’ = 0x6C
- ‘o’ = 0x6F
ASCII ...

- the string “Hello”, if at address 0x1000, stored as follows

<table>
<thead>
<tr>
<th>H</th>
<th>e</th>
<th>l</th>
<th>l</th>
<th>o</th>
<th>NUL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x48</td>
<td>0x65</td>
<td>0x6c</td>
<td>0x6c</td>
<td>0x6f</td>
<td>0x00</td>
</tr>
<tr>
<td>0x1000</td>
<td>0x1001</td>
<td>0x1002</td>
<td>0x1003</td>
<td>0x1004</td>
<td>0x1005</td>
</tr>
</tbody>
</table>

- ASCII strings early always NUL terminated
- only 96 ASCII characters are printable, the remainder are control codes
- example control codes

| 0x0A | LF | line feed |
| 0x0D | CR | carriage return |
| 0x08 | BS | backspace |
| 0x09 | HT | horizontal tab |
| 0x1B | ESC | escape |
| 0x00 | NUL | NUL |
Example

- copy a NULL terminated ASCII string from 0x1000 (in read-only flash memory) to 0x40000000 (RAM)

```
LDR   R1, =0x1000  ; R1 -> src
LDR   R2, =0x40000000  ; R2 -> dst
L    LDRB  R0, [R1]  ; get next char from src string
    STRB  R0, [R2]  ; store char in dst string
    ADD  R1, R1, #1  ; move to next src char
    ADD  R2, R2, #1  ; move to next dst char
    CMP  R0, #0  ; if not finished (char = 0x00) ...
    BNE  L  ; copy next character
```