CS4803DGC Design Game Console

Spring 2010
Prof. Hyesoon Kim
DevKit Pro & libnds

- DevKit Pro is a collection of tool chain for homebrew applications developers for various architectures
- DevKitARM: ARM binaries
- Not official development tool chain
  - Much simpler and naïve
- libnds
  - Started with header files for definition
  - Extended to have other data structures, simple APIs
- *.nds
  - A binary for Nintendo DS, a separate region for ARM7 and ARM9

int main(void) {

consoleDemoInit(); //Initialize the console

irqSet(IRQ_VBLANK, Vblank); //this line says: When the IRQ_VBLANK interrupt occurs execute function Vblank
iprintf("Hello DS dev'rs\n");

while(1) {
    printf("\x1b[10;0HFrame = %d",frame); //print out the current frame number
    swiWaitForVBlank(); //This line basically pauses the while loop and makes it
    //wait for the IRQ_VBLANK interrupt to occur. This way, we print only once
    //per frame.
}
return 0;
}
Framebuffer Mode

- 2 screens, 2 GPUs, only bottom has a touch screen
- A screen mode where the screen is mapped to a portion of memory
- Writing data to this memory area will result in data appearing on the screen
- Each screen’s pixel is represented by 2 B.
- Represented with 555 format
- 0123 4567 0123 4567
  - rrr rr-- ---- ---- (bitmask: 0x7C00)
  - ---- --gg ggg- ---- (bitmask: 0x3E0)
  - ---- ---- --b bbbb (bitmask: 0x1F)
- But a simple macro
  RGB15 Color
  RGB15(31,0,0) Red
  RGB15(0,31,0) Green
  RGB15(0,0,31) Blue
  RGB15(0,0,0) Black
  RGB15(31,31,31) White
Vertical Blank Interrupt

Every 1/60th seconds, the hardware redraws.
Visiting each pixel row by row, copying the contents of the framebuffer for that pixel to the hardware screen pixel
Vertical blank interrupt: when it finishes drawing the screen.
Assignment #6

• ARM assembly code
  – Build a simple counter
  – A key increment a counter
  – B key decrement a counter
  – Start: reset to zero
  – Up arrow +10
  – Down arrow -10
  – No need to use interrupt, use a polling method
GCC Inline Assembly Programming

• Instead of pure assembly coding, we will use inline assembly programming
• Not only ARM, x86 etc.
• Good place to look at
  http://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html#ss5.3

NOP
asm( "mov r0, r0
  "mov r0, r0
  "mov r0, r0
  "mov r0, r0
  "mov r0, r0" );

Use delimiters Linefeed or tab to differentiate assembly lines
ASM Examples

asm(code :
   output operand list : /* optional*/
   input operand list : /* optional*/
   clobber list /* optional*/
);
/* Rotating bits example */
asm("mov %[result], %[value], ror #1" :
   [result] "=r" (y) :
   [value] "r" (x));
Symbolic name encoded in square brackets followed by a constraint string, followed by a C expression enclosed in parentheses
 e.g.) sets the current program status register of the ARM CPU
asm("msr cpsr,%[ps]" :
   [ps]"r"(status)
);
Clobber List

- Some instructions clobber some hardware registers.
- We have to list those registers in the clobber-list
- Shouldn’t list input & output (already given)
Example: Simple ADD and Print

```c
int main(void) {
    //consoleDemoInit();
    int* notGood= (int *)0xb0; //bad
    *notGood= 10;
    int better=20;
    irqSet(IRQ_VBLANK, Vblank);
    printf("Hello CS4803DGC");
    // case 1
    asm("MOV R1, #0xb0"); //init R1 to address
    asm("LDR R0, [R1]");
    asm("ADD R0, R0, R0");
    asm("STR R0, [R1]");
    // case 2
    asm ("MOV R1, %[value]::[value]"r"(better));
    asm ("ADD R1, R1, R1");
    asm ("MOV %[result], R1":[result]"=r"(better):);
    while(1) {
        swiWaitForVBlank();
        printf("\x1b[10;0HFrame = %d",frame);
        printf ("\nbah is: %d, %d", *notGood, better);
    }
    return 0;
}
```

Please note that this code does not correctly!
# Nintendo DS Input System

- Button, touch screen, microphone
- Libnds key definition

<table>
<thead>
<tr>
<th>Key Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_A</td>
<td>1 &lt;&lt; 0</td>
<td>A Button</td>
</tr>
<tr>
<td>KEY_B</td>
<td>1 &lt;&lt; 1</td>
<td>B Button</td>
</tr>
<tr>
<td>KEY_SELECT</td>
<td>1 &lt;&lt; 2</td>
<td>Select Button</td>
</tr>
<tr>
<td>KEY_START</td>
<td>1 &lt;&lt; 3</td>
<td>Start Button</td>
</tr>
<tr>
<td>KEY_RIGHT</td>
<td>1 &lt;&lt; 4</td>
<td>Right D-pad</td>
</tr>
<tr>
<td>KEY_LEFT</td>
<td>1 &lt;&lt; 5</td>
<td>Left D-pad</td>
</tr>
<tr>
<td>KEY_UP</td>
<td>1 &lt;&lt; 6</td>
<td>Up D-pad</td>
</tr>
<tr>
<td>KEY_DOWN</td>
<td>1 &lt;&lt; 7</td>
<td>Down D-pad</td>
</tr>
<tr>
<td>KEY_R</td>
<td>1 &lt;&lt; 8</td>
<td>R Button</td>
</tr>
<tr>
<td>KEY_L</td>
<td>1 &lt;&lt; 9</td>
<td>L Button</td>
</tr>
<tr>
<td>KEY_X</td>
<td>1 &lt;&lt; 10</td>
<td>X Button</td>
</tr>
<tr>
<td>KEY_Y</td>
<td>1 &lt;&lt; 11</td>
<td>Y Button</td>
</tr>
<tr>
<td>KEY_TOUCH</td>
<td>1 &lt;&lt; 12</td>
<td>Pen Touching Screen (no coordinates)</td>
</tr>
<tr>
<td>KEY_LID</td>
<td>1 &lt;&lt; 13</td>
<td>Lid shutting (useful for sleeping)</td>
</tr>
</tbody>
</table>
Memory Mapped I/O
The current status of the keys is stored in memory at address 0x4000130.

When no key is pressed - the value is 1023.

A key press causes a change in the value at this location. The new value depends on which key is pressed.

Here are the values for various keys.

A- #1022  b 11 1111 1110
B- #1021  b 11 1111 1101
start- #1015  b 11 1111 1011
UP- #959   b 11 1011 1111
DOWN- #895  b 11 0111 1111
Example: Reading Keys

asm ("MOV R4, #0x0000"); //R4 has the counter.. funny things happening with R1
   while(1) {
   swiWaitForVBlank();
      //init R4 to address
   asm ("MOV R0, #0x4000000"); //R0 has the address
   asm ("ADD R0, #0x130"); // finished moving address

   //load value from that address
   asm ("LDR R2, [R0]");

   // check the register value of R2 and compare and then increment the counter
   // use condition code or shift etc.

   //move counter value from R2 to C variable
   asm ("MOV %[result], R2":[result]"=r"(result_):);
Caution!

- Compiler still rearranges the assembly code.
- Use ASM `volatile` (""") to prevent compiler’s optimizations.
- Default compilation mode is ARM-thumb.
- The makefile has to be modified- set it to no optimization by `-O0`
- change line ARCH := -mthumb -mthumb-interwork TO ARCH := -marm
ARM ASSEMBLY PROGRAMMING
## Control Flow Instructions

<table>
<thead>
<tr>
<th>Return Instruction</th>
<th>Previous State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ARM R14&lt;sub&gt;x&lt;/sub&gt;</td>
</tr>
<tr>
<td>BL</td>
<td>PC + 4</td>
</tr>
<tr>
<td>SWI</td>
<td>PC + 4</td>
</tr>
<tr>
<td>UDEF</td>
<td>PC + 4</td>
</tr>
<tr>
<td>FIQ</td>
<td>PC + 4</td>
</tr>
<tr>
<td>IRQ</td>
<td>PC + 4</td>
</tr>
<tr>
<td>PABT</td>
<td>PC + 4</td>
</tr>
<tr>
<td>DABT</td>
<td>PC + 8</td>
</tr>
<tr>
<td>RESET</td>
<td>-</td>
</tr>
</tbody>
</table>
## Instruction Format

<table>
<thead>
<tr>
<th>Cond</th>
<th>Opcode</th>
<th>S</th>
<th>Rn</th>
<th>Rd</th>
<th>Operand2</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>000000</td>
<td>0</td>
<td>Rn</td>
<td>Rd</td>
<td>Rs 1001 Rm</td>
</tr>
<tr>
<td>0001</td>
<td>000100</td>
<td>111111110001</td>
<td>Rn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0002</td>
<td>000001</td>
<td>1</td>
<td>S H 1</td>
<td>Offset</td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>000000</td>
<td>1</td>
<td>Rn</td>
<td>Rd</td>
<td>Offset</td>
</tr>
<tr>
<td>011</td>
<td>000000</td>
<td>1</td>
<td>Rn</td>
<td>Rd</td>
<td>Offset</td>
</tr>
<tr>
<td>100</td>
<td>000000</td>
<td>1</td>
<td>Rn</td>
<td>Rd</td>
<td>Offset</td>
</tr>
<tr>
<td>101</td>
<td>000000</td>
<td>1</td>
<td>CRd</td>
<td>CP#</td>
<td>Offset</td>
</tr>
<tr>
<td>110</td>
<td>000000</td>
<td>1</td>
<td>CRn</td>
<td>CRd</td>
<td>Offset</td>
</tr>
<tr>
<td>111</td>
<td>000000</td>
<td>1</td>
<td>CRn</td>
<td>CRd</td>
<td>Offset</td>
</tr>
</tbody>
</table>

### Data/Processing/PSR Transfer
- Multiply
- Multiply Long
- Single Data Swap
- Branch and Exchange
- Halfword Data Transfer: register offset
- Halfword Data Transfer: immediate offset
- Single Data Transfer
- Undefined
- Block Data Transfer
- Branch
- Coprocessor Data Transfer
- Coprocessor Data Operation
- Coprocessor Register Transfer
- Software Interrupt

---

![Georgia Tech College of Computing logo](https://www.gatech.edu/)
Condition Code

- **N**: Negative (the last ALU operation)
- **Z**: zero (the last ALU operation)
- **C**: carry (the last ALU or from shifter)
- **V**: overflow

Steve Furber, ARM system-on-chip architecture 2nd edition
## ARM condition codes

Table 5.3  ARM condition codes.

<table>
<thead>
<tr>
<th>Opcode [31:28]</th>
<th>Mnemonic extension</th>
<th>Interpretation</th>
<th>Status flag state for execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>EQ</td>
<td>Equal / equals zero</td>
<td>Z set</td>
</tr>
<tr>
<td>0001</td>
<td>NE</td>
<td>Not equal</td>
<td>Z clear</td>
</tr>
<tr>
<td>0010</td>
<td>CS/HS</td>
<td>Carry set / unsigned higher or same</td>
<td>C set</td>
</tr>
<tr>
<td>0011</td>
<td>CC/LO</td>
<td>Carry clear / unsigned lower</td>
<td>C clear</td>
</tr>
<tr>
<td>0100</td>
<td>MI</td>
<td>Minus / negative</td>
<td>N set</td>
</tr>
<tr>
<td>0101</td>
<td>PL</td>
<td>Plus / positive or zero</td>
<td>N clear</td>
</tr>
<tr>
<td>0110</td>
<td>VS</td>
<td>Overflow</td>
<td>V set</td>
</tr>
<tr>
<td>0111</td>
<td>VC</td>
<td>No overflow</td>
<td>V clear</td>
</tr>
<tr>
<td>1000</td>
<td>HI</td>
<td>Unsigned higher</td>
<td>C set and Z clear</td>
</tr>
<tr>
<td>1001</td>
<td>LS</td>
<td>Unsigned lower or same</td>
<td>C clear or Z set</td>
</tr>
<tr>
<td>1010</td>
<td>GE</td>
<td>Signed greater than or equal</td>
<td>N equals V</td>
</tr>
<tr>
<td>1011</td>
<td>LT</td>
<td>Signed less than</td>
<td>N is not equal to V</td>
</tr>
<tr>
<td>1100</td>
<td>GT</td>
<td>Signed greater than</td>
<td>Z clear and N equals V</td>
</tr>
<tr>
<td>1101</td>
<td>LE</td>
<td>Signed less than or equal</td>
<td>Z set or N is not equal to V</td>
</tr>
<tr>
<td>1110</td>
<td>AL</td>
<td>Always</td>
<td>any</td>
</tr>
<tr>
<td>1111</td>
<td>NV</td>
<td>Never (do not use!)</td>
<td>none</td>
</tr>
</tbody>
</table>