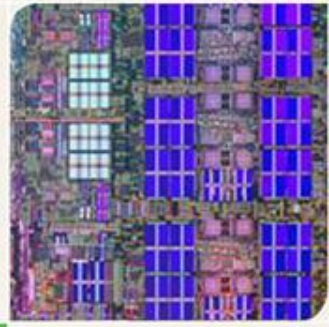
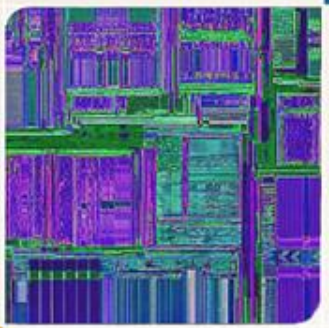
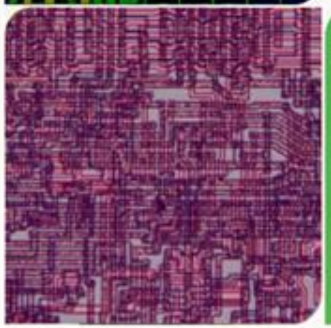



CS4803DGC Design Game Console

Spring 2010

Prof. Hyesoon Kim

Georgia Tech College of Computing

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

http://patater.com/files/projects/manual/manual.html#id2612503

Review Hello World

```
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) {
    iprintf("\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

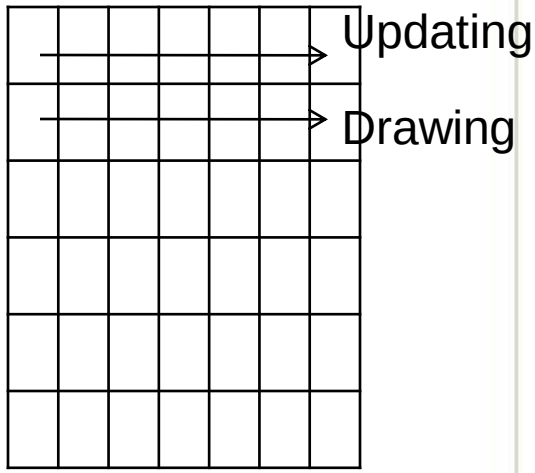
Vertical Blank Interrupt

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

Interrupt





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

http://www.ethernut.de/en/documents/arm-inline-asm.html

NOP

```
asm( "mov r0, r0\n\t"
     "mov r0, r0\n\t"
      "mov r0, r0\n\t"
       "mov r0, r0" );
```

Use deliminaters Linefeed or tab to differentitate 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

```
nt 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();
                // print at using ansi escape sequence \x1b[line;columnH
                printf("\x1b[10;0HFrame = %d",frame);
                printf ("\nblah 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

KEY_A	1 << 0	A Button
KEY_B	1 << 1	B Button
KEY_SELECT	1 << 2	Select Button
KEY_START	1 << 3	Start Button
KEY_RIGHT	1 << 4	Right D-pad
KEY_LEFT	1 << 5	Left D-pad
KEY_UP	1 << 6	Up D-pad
KEY_DOWN	1 << 7	Down D-pad
KEY_R	1 << 8	R Button
KEY_L	1 << 9	L Button
KEY_X	1 << 10	X Button
KEY_Y	1 << 11	Y Button
KEY_TOUCH	1 << 12	Pen Touching Screen (no coordinates)
KEY_LID	1 << 13	Lid shutting (useful for sleeping)

Memory Mapped I/O

0x4000130	

Key Mapping in Nintendo DS

- 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



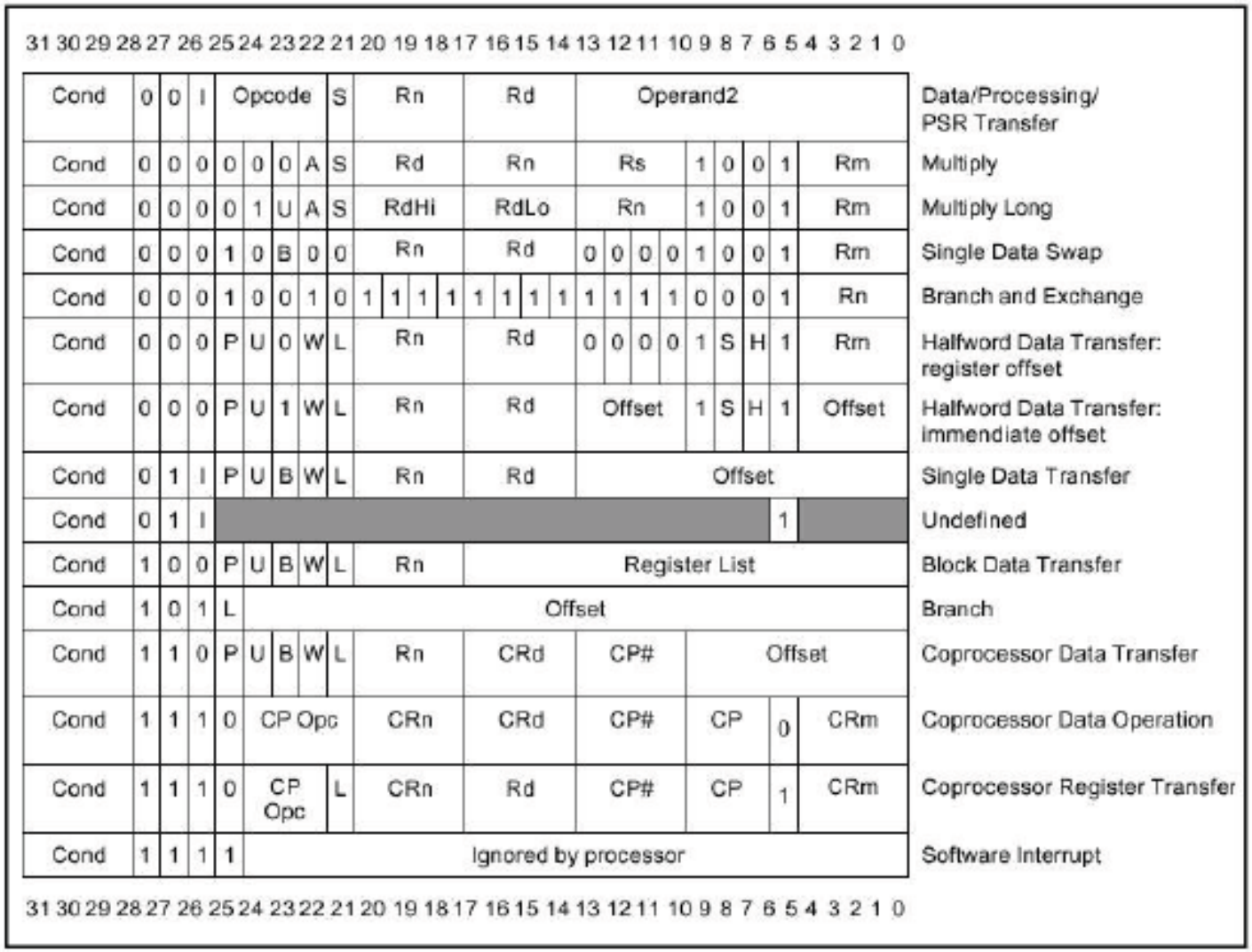
ARM ASSEMBLY PROGRAMMING



Control Flow Instructions

	Return Instruction	Previous State	
		ARM R14_x	THUMB R14_x
BL	MOV PC, R14	PC + 4	PC + 2
SWI	MOVS PC, R14_svc	PC + 4	PC + 2
UDEF	MOVS PC, R14_und	PC + 4	PC + 2
FIQ	SUBS PC, R14_fiq, #4	PC + 4	PC + 4
IRQ	SUBS PC, R14_irq, #4	PC + 4	PC + 4
PABT	SUBS PC, R14_abt, #4	PC + 4	PC + 4
DABT	SUBS PC, R14_abt, #8	PC + 8	PC + 8
RESET	NA	-	-

Instruction Format



Condition Code

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

ARM condition codes

Table 5.3 *ARM condition codes.*

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

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