ECE 471 – Embedded Systems
Lecture 8

Vince Weaver

http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

24 September 2015

Announcements

- HW#4 will be posted soon
- Will require an LED, a breadboard, and some jumper wires. I handed out some GPIO wires in class.

THUMB-2 Wrapup

- Extension of THUMB to have both 16-bit and 32-bit instructions
- 32-bit instructions *not* standard 32-bit ARM instructions. It's a new encoding that allows an instruction to be 32-bit if needed.
- Most 32-bit ARM instructions have 32-bit THUMB-2 equivalents *except* ones that use conditional execution. The `it` instruction was added to handle this.
- `rsc` (reverse subtract with carry) removed

- Shifts in ALU instructions are by constant, cannot shift by register like in arm32
- THUMB-2 code can assemble to either ARM-32 or THUMB2
  The assembly language is compatible.
  Common code can be written and output changed at time of assembly.
- Instructions have "wide" and "narrow" encoding.
  Can force this (`add.w` vs `add.n`).
- Need to properly indicate "s" (set flags).
  On regular THUMB this is assumed.

THUMB-2 Coding

- See `test_thumb2.s`
- Use `.syntax unified` at beginning of code
- Use `.arm` or `.thumb` to specify mode

New THUMB-2 Instructions

- BFI – bit field insert
- RBIT – reverse bits
- movw/movt – 16 bit immediate loads
- TB – table branch
- IT (if/then)
- cbz – compare and branch if zero; only jumps forward

Thumb-2 12-bit immediates

```
top 4 bits 0000 -- 00000000 00000000 00000000 abcdefgh
           0001 -- 00000000 abcdefgh 00000000 abcdefgh
           0010 -- abcdefgh 00000000 abcdefgh 00000000
           0011 -- abcdefgh abcdefgh abcdefgh abcdefgh
           0100 -- 1bcdedfh 00000000 00000000 00000000
            ...
           1111 -- 00000000 00000000 00000001 bcdefgh0
```

Compiler

- Original RASPBERRY PI DOES NOT SUPPORT THUMB2
- `gcc -S hello_world.c`
  By default is arm32
- `gcc -S -march=armv5t -mthumb hello_world.c`
  Creates THUMB (won't work on Raspberry Pi due to HARDFP arch)
- `-mthumb -march=armv7-a` Creates THUMB2

IT (If/Then) Instruction

- Allows limited conditional execution in THUMB-2 mode.
- The directive is optional (and ignored in ARM32) the assembler can (in-theory) auto-generate the IT instruction
- Limit of 4 instructions

Example Code

```
it cc
addcc r1,r2
```

```
itete cc
addcc r1,r2
addcs r1,r2
addcc r1,r2
addcs r1,r2
```

`ll` Example Code

```
        ittt cs @ If CS Then Next plus CS for next 3
discrete_char:
        ldrbcs  r4,[r3]      @ load a byte
        addcs   r3,#1        @ increment pointer
        movcs   r6,#1        @ we set r6 to one so byte
        bcs.n   store_byte   @ and store it
offset_length:
```

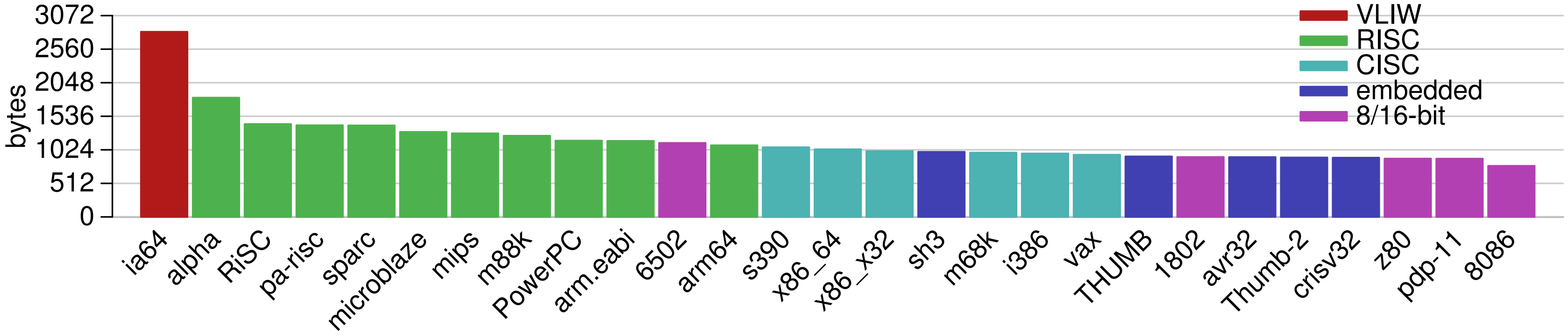
AARCH64

- 32-bit fixed instruction encoding
- 31 64-bit GP registers (x0-x30), zero register (x30)
- PC is not a GP register
- only branches conditional
- no load/store multiple
- No thumb

Code Density

- Overview from my `ll` ICCD'09 paper
- Show code density for variety of architectures, recently added Thumb-2 support.
- Shows overall size, though not a fair comparison due to operating system differences on non-Linux machines

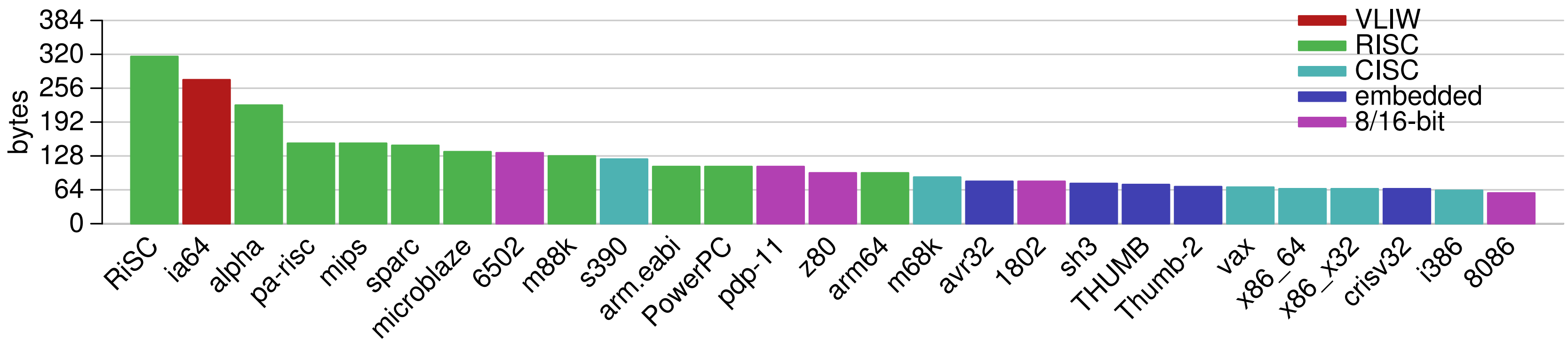
Code Density – overall



lzss compression

- Printing routine uses lzss compression
- Might be more representative of potential code density

Code Density – lzss



Coding Directly for the Hardware

One way of developing embedded systems is coding to the raw hardware, as you did with the STM Discovery Boards in ECE271.

- Compile code
- Prepare for upload (hexbin?)
- Upload into FLASH
- Boots to offset

- Setup, flat memory (usually), stack at top, code near bottom, IRQ vectors

- Handle Interrupts

- Must do I/O directly (no drivers)
  Although if lucky, can find existing code.

- **Code is specific to the hardware you are on**

Instead, one can use an Operating System

Why Use an Operating System?

- Provides Layers of Abstraction
  - Abstract hardware: hide hardware differences. same hardware interface for classes of hardware (things like video cameras, disks, keyboards, etc) despite differing implementation details
  - Abstract software: with VM get linear address space, same system calls on all systems
  - Abstraction comes at a cost. Higher overhead, unknown timing

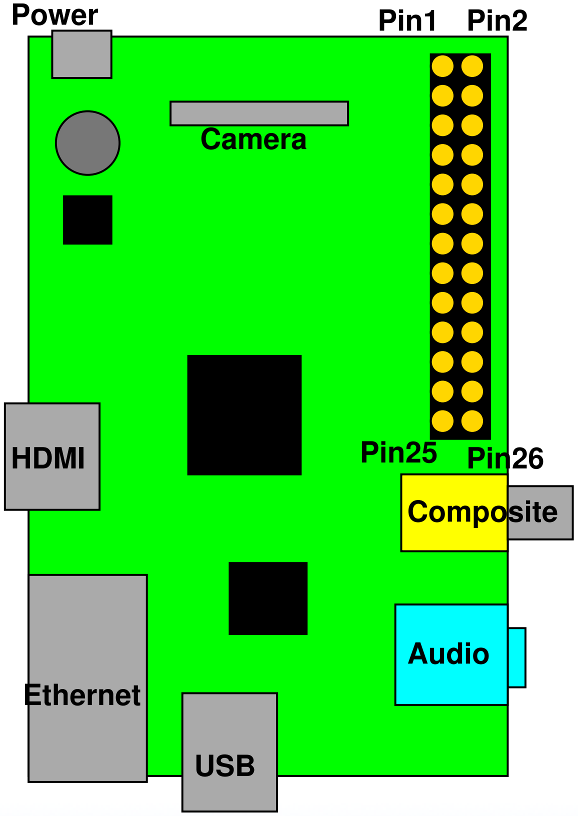
- Multi-tasking / Multi-user
- Security, permissions (Linus dial out onto /dev/hda)
- Common code in kernel and libraries, no need to re-invent

What's included with an OS

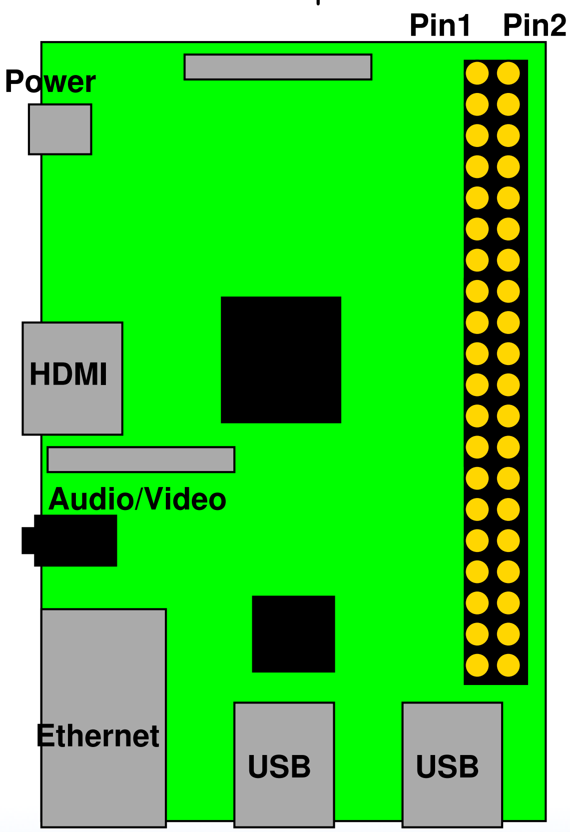
- kernel / drivers – Linux definition
- also system libraries – Solaris definition
- low-level utils / software / GUI – Windows definition
  Web Browser included?
- Linux usually makes distinction between the OS Kernel and distribution. OSX/Windows usually doesn't.

Brief Overview of the Raspberry Pi Board

Model B



Model B+



Rasp-pi Header

- Model B has 17 GPIOs (out of 26 pins), B+ has 9 more (out of 40)
- 3.3V signaling logic. Need level shifter if want 5V or 1.8V
- Linux by default configures some for other purposes (serial, i2c, SPI)

Rasp-pi Header

3.3V	1	2	5V
GPIO2 (SDA)	3	4	5V
GPIO3 (SCL)	5	6	GND
GPIO4	7	8	GPIO14 (UART_TXD)
GND	9	10	GPIO15 (UART_RXD)
GPIO17	11	12	GPIO18 (PCM_CLK)
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10 (MOSI)	19	20	GND
GPIO9 (MISO)	21	22	GPIO25
GPIO11 (SCLK)	23	24	GPIO8 (CE0)
GND	25	26	GPIO7 (CE1)

ID_SD (EEPROM)	27	28	ID_SC (EEPROM)
GPIO5	29	30	GND
GPIO6	31	32	GPIO12
GPIO13	33	34	GND
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
GND	39	40	GPIO21

How you enable GPIO on STM32L

A lot of read/modify/write instructions to read current register values and then to shift/mask to write out updated bitfields.

- Enable GPIO Clock
- Set output mode for GPIO.
- Set GPIO type.
- Set pin clock speed.
- Set pin pull-up/pull-down
- Set or clear GPIO pin.

How you enable on BCM2835 (Rasp-pi)

- Documented in BCM2835 ARM Peripherals Manual
- 53 GPIOs (not all available on board)
- Similar to how done on STM32L... but we have an operating system

Letting the OS handle it for you

Linux GPIO interface

- `Documentation/gpio/sysfs.txt`
- sysfs and string based

A few low-level Linux Coding Instructions

Enable a GPIO for use

To enable GPIO 17:

write “17” to /sys/class/gpio/export

To disable GPIO 17:

write “17” to /sys/class/gpio/unexport

```
char buffer[10];
fd=open("/sys/class/gpio/export",O_WRONLY);
if (fd<0) fprintf(stderr,"\tError enabling\n");
strcpy(buffer,"17");
write(fd,buffer,2);
close(fd);
```

Set GPIO Direction

To make GPIO 17 an input:

write "in" to /sys/class/gpio/gpio17/direction

To make GPIO 17 an output:

write "out" to /sys/class/gpio/gpio17/direction

```
fd=open("/sys/class/gpio/gpio17/direction",O_WRONLY);
if (fd<0) fprintf(stderr,"Error!\n");
write(fd,"in",2);
close(fd);
```

Read/Write GPIO Value

To read value of GPIO 17:
read `/sys/class/gpio/gpio17/value`
To write value of GPIO 17:
write `/sys/class/gpio/gpio17/value`

```
fd=open("/sys/class/gpio/gpio17/value",O_WRONLY);
if (fd<0) fprintf(stderr,"Error␣writing!\n");
write(fd,"1",1);
close(fd);
```

Note, if reading and you do not close after read you will have to rewind using `lseek(fd,0,SEEK_SET);` after your read.

Delay

- Busy delay (like in 271).
  `for(i=0;i<1000000;i++);`
  Harder to do in C. Why?
  Compiler optimizes away.

- `usleep()` puts process to sleep for a number of microseconds. But can have issues if want exact delay. Why? OS potentially context switches every 100ms.

- Other ways to implement: Set up PWM? Timers?

Implementations

- Busy loop. Bad, burns CPU / power
- usleep(). But can take a while to respond.
- Interrupt when ready! poll()

GPIO Interrupts on Linux

May need a recent version of Raspbian.

First write "rising", "falling", or "both" to `/sys/class/gpio/gpio17/edge`.

Then open and poll `/sys/class/gpio/gpio17/value`.

```
struct pollfd fds;
int result;

fd=open("/sys/class/gpio/gpio18/value",O_RDONLY);
fds.fd=fd;
fds.events=POLLPRI|POLLERR;
while(1) {
    result=poll(&fds,1, -1);
    if (result<0) printf("Error!\n");
    lseek(fd,0,SEEK_SET);
read(fd,buffer,1); }
```

Debouncing!

- Pull-up / Pull-down resistor. Why?
- Noisy switches, have to debounce
- Manual, no built-in debounce like on STM32L

Why Use an Operating System?

- Provides Layers of Abstraction
  - Abstract hardware: hide hardware differences. same hardware interface for classes of hardware (things like video cameras, disks, keyboards, etc) despite differing implementation details
  - Abstract software: with VM get linear address space, same system calls on all systems
  - Abstraction comes at a cost. Higher overhead, unknown timing

- Multi-tasking / Multi-user
- Security, permissions (Linus dial out onto /dev/hda)
- Common code in kernel and libraries, no need to re-invent

What's included with an OS

- kernel / drivers – Linux definition
- also system libraries – Solaris definition
- low-level utils / software / GUI – Windows definition
  Web Browser included?
- Linux usually makes distinction between the OS Kernel and distribution. OSX/Windows usually doesn't.