

CSCE 462

SOC:

- System On a Chip

Latency vs Throughput

- latency: time to complete one operation
- throughput: operations per second
- memory has both latency and throughput

GPIO

- General Purpose Input/Output
- sometimes called “Port A”

Raspberry Pi 3

- 1.2GHz 4-core processor
- 1GB RAM
- 40 board pins, 26 are GPIO
- WiFi, Ethernet

positive vs negative logic

- positive logic
 - AKA pull down resistor
 - switch:
 - * resistor from GPIO to ground
 - * switch from Vcc to GPIO
 - LED: resistor and LED in series between GPIO and ground
- negative logic
 - AKA pull up resistor
 - switch:
 - * resistor between GPIO and Vcc
 - * switch between GPIO and ground
 - LED: resistor and LED in series between GPIO and Vcc

- in all cases, LED must be oriented correctly (pointing toward lower voltage)
- GPIO should only be used for less than 8mA
 - if LED or something needs more than that, connect it to an inverter, then connect the LED using the other kind of logic (positive/negative)
- typically use about 10k Ohm resistor with switches

ARM

- link register: stores return address of function
 - means that leaf function calls are faster
- flag bits
 - **c** carry (unsigned overflow)
 - * **c** set after unsigned addition => overflow
 - * **c NOT** set after unsigned subtraction => overflow
 - **v** overflow (signed overflow)
 - * **v** set after signed addition => overflow
 - * **v** set after signed subtraction => overflow
 - **z** zero
 - **n** negative
 - **p** parity
 - * just xor of all bits together
 - flags set after every instruction?
- <http://www.peter-cockerell.net/aalp/html/ch-2.html>
- <http://www.peter-cockerell.net/aalp/html/ch-3.html>
- instruction format: **add{cond}{s} <dest>, <lhs>, <rhs>**
 - **s**, if present, means the instruction can change processor flags (e.g. over/underflow)
- **ldr**: load word (to register)
- **str**: store word
 - load/store can have suffix to do non-32-bit sizes:
 - * **b**: byte
 - * **h**: half-word (16 bits)
 - **ldr Rd, =label** (pseudo-instruction): loads address of label into register
- 16 registers: R0-R15, and CPSR
 - R0-R3: arguments to function call, and return value
 - R12 IP: Intra procedural call
 - R13 SP: Stack Pointer
 - R14 LR: Link Register
 - * stores return address
 - R15 PC: Program Counter
 - * you can read/write to this directly, it's not special (though not a good idea)
 - CPSR: Current Program State Register:

- * stores flags about the state of the program: negative, zero, carry, overflow, underflow, privileged mode, etc...
- * can be modified directly (want to do read-modify-write state backup first)
- R4–R11 and R13 SP are callee preserved: function should push during prologue and pop in epilogue of function
- callee must preserve R14 LR if it wishes to call subroutines
- R12 IP is weird. Used for libraries as stack space?
- push/pop for accessing stack
 - you can push/pop multiple things at once, but you need to reverse at the end:


```
push {ip, lr, sp}
pop {sp, lr, ip}
```
- bl: branch and link (set link register). use to call subroutines
- bx lr: branch to link register, to return from subroutine
- constants: `mov r1, =label`
 - can do `ldr r0, =0x523` (translates to PC relative load)
- literal: `mov r0, #5`, `mov r2, #0x1C`
- memory access: `ldr r0 [r0]`, `str r0 [r1, #offset]`
 - memory access is sometimes done relative to PC

powers of two:

- 2^{10} is 1K
- 2^{20} is 1M
- 2^{30} is 1G

SysTick timer

- 24-bit timer that counts down at bus clock frequency
- only counts down
- control registers (memory mapped)
 - `ctrl`: control
 - * can have different sources (but is always internal on raspberry pi)
 - * bit 16: COUNT bit
 - 1 if current value has been 0 since the last time you read ctrl register
 - 0 otherwise
 - * enable bit to start timer (bit 0)
 - `reload`: value to start at (after counting wraps)
 - `current`: current value timer is counting at

- * whenever you write to this, no matter what you write, it resets the counter

Finite State Machine (FSM)

- Mealy vs Moore
 - Moore
 - * output on states
 - * output based on current state only
 - Mealy
 - * output on state transitions (edges of graph)
 - * output based on state and current input
- can have wait time at each state
- can be implemented like a linked list (or other ways)
- implementation
 - linked list: pointers link to adjacent nodes
 - table: stores indexes of adjacent nodes
- if you implement with a data structure, you need a driver function to advance states and whatnot
- data structure implementation seems way over complicated

Software abstraction

- Define a problem with a minimal set of basic, abstract principles / concepts
- Separation of concerns via interface/policy mechanisms
- Straightforward, mechanical path to implementation
- advantages
 - faster to develop
 - easier to debug
 - easier to change