ENGR-304-L

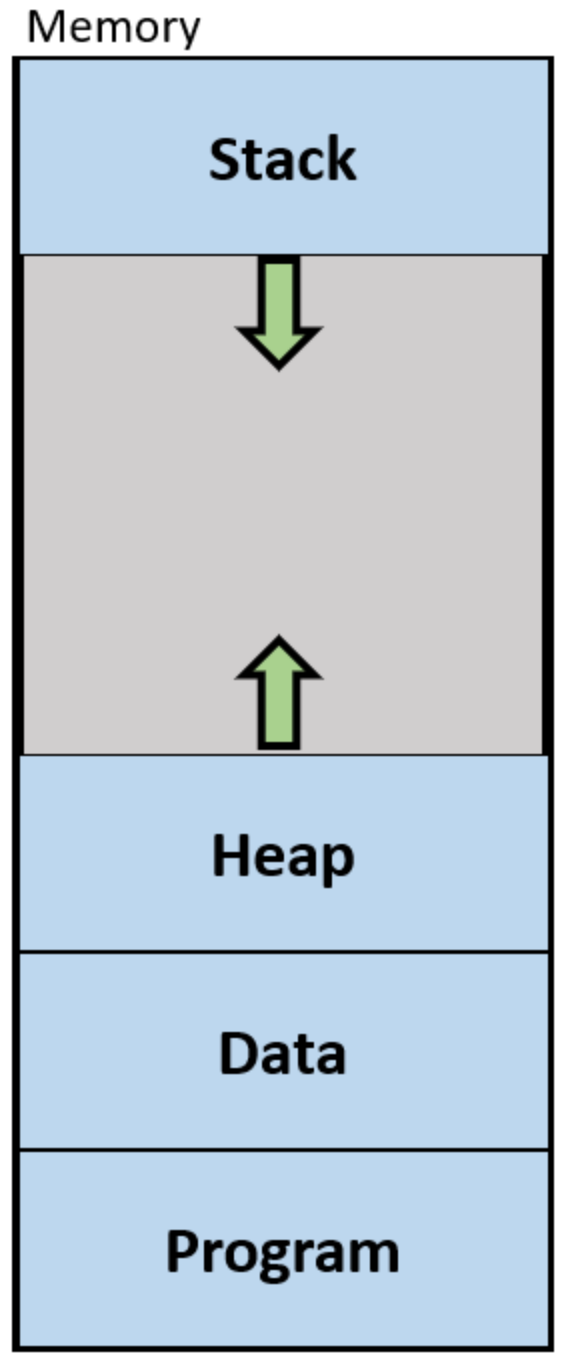
Software Lab 05

Agenda

1. Attendance
2. Recap
3. Masking
4. IO Devices
5. Assembly with IO Devices
6. Getting Started

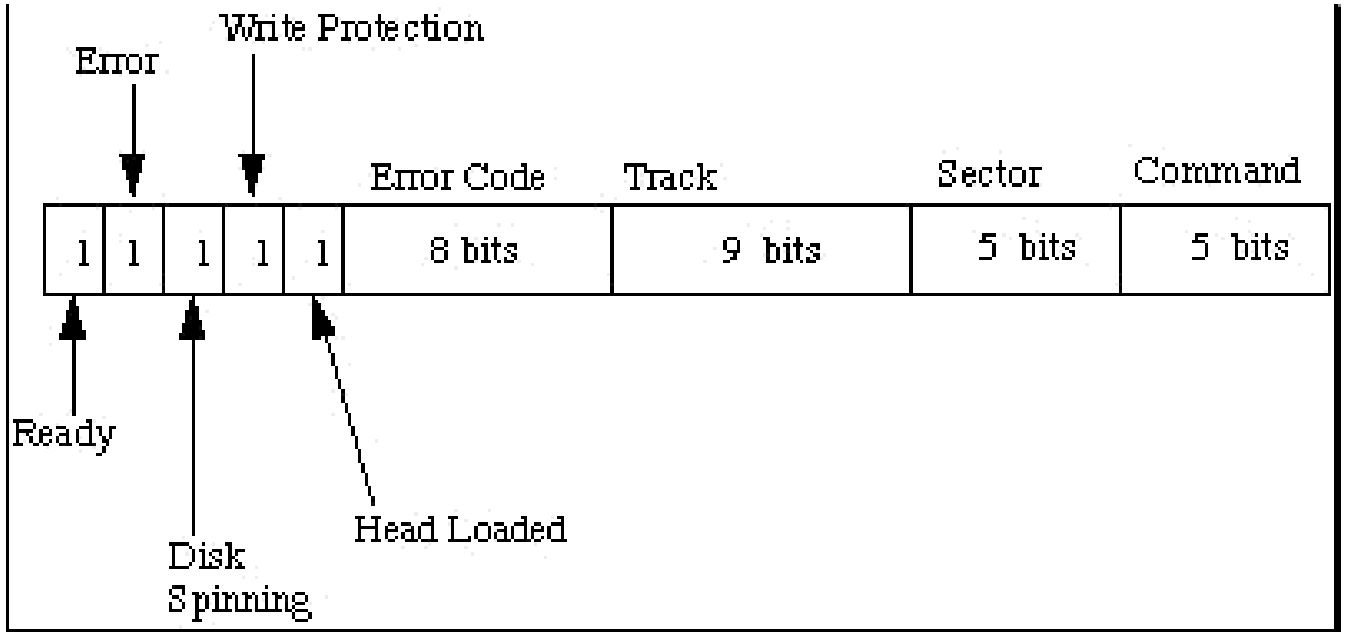
Recap

- The stack is memory used to protect values of registers when function calls are made
- The stack grows towards program and data memory



Bit-Masks

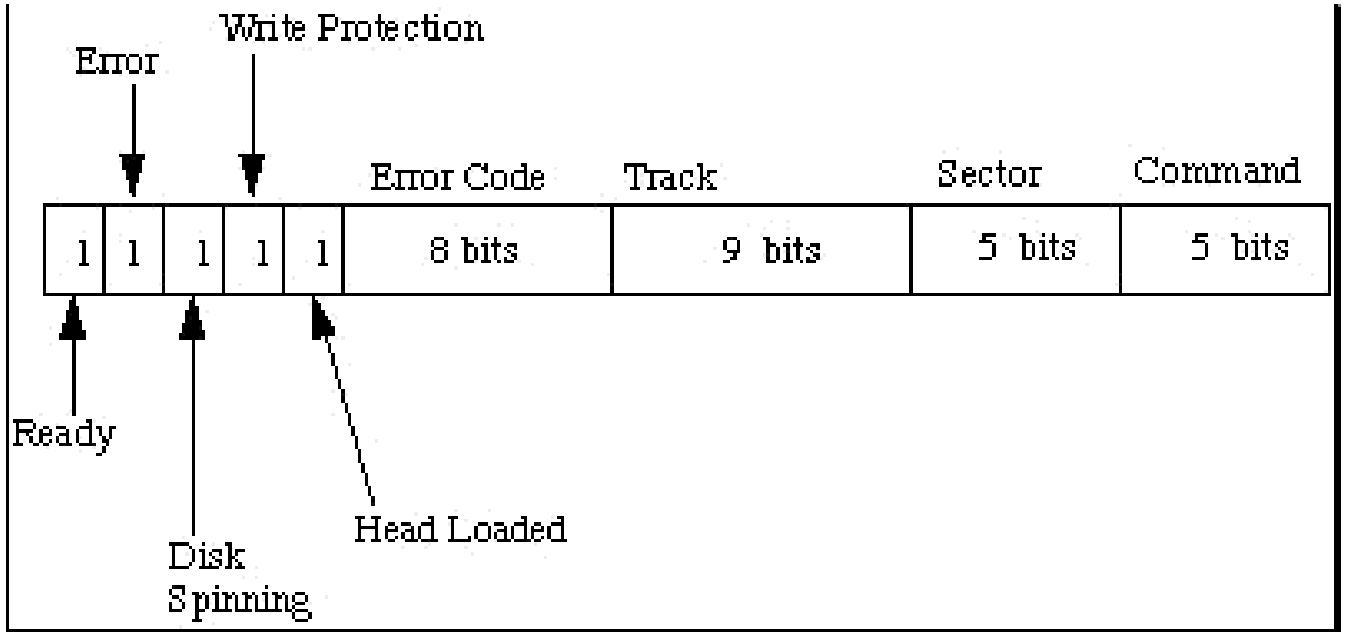
- Recall protocols & bit-fields



https://users.cs.cf.ac.uk/Dave.Marshall/C/drive.gif

Bit-Masks

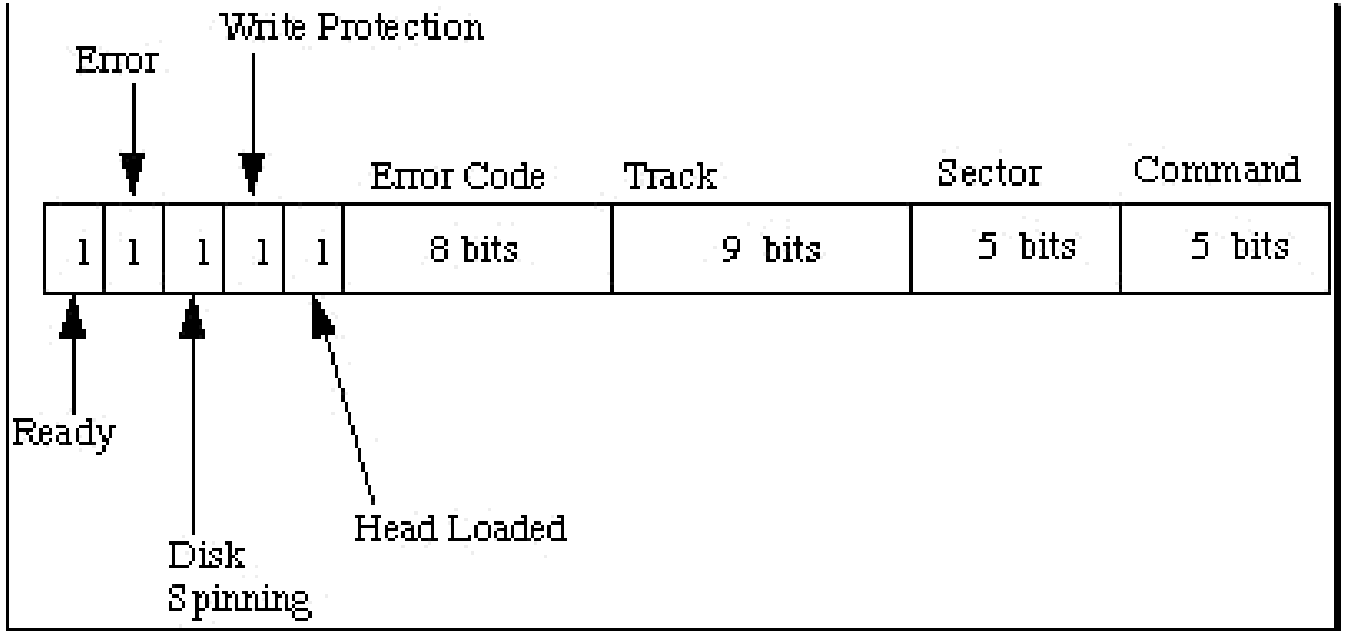
- Recall protocols & bit-fields
- How do we get just the "Write Protection" bit?



https://users.cs.cf.ac.uk/Dave.Marshall/C/drive.gif

Bit-Masks

- How do we get just the "Write Protection" bit?

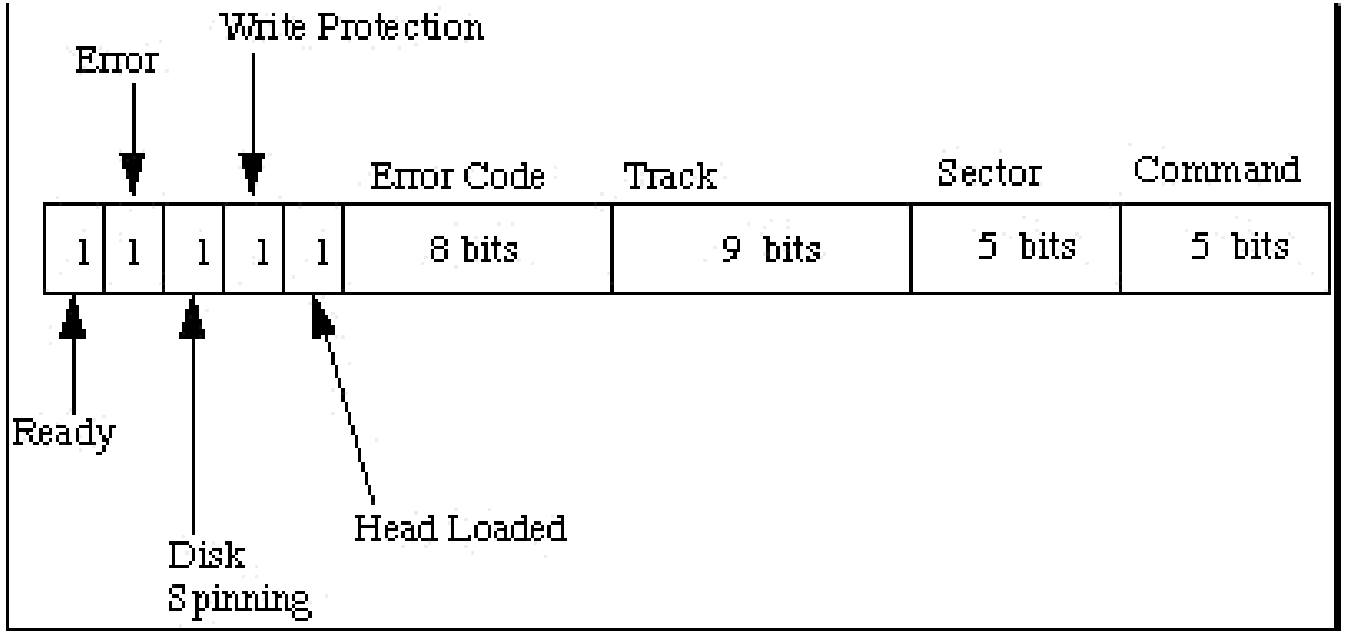



```
DATA:·····0b1_0_1_1_0_00011100_110000000_00101_01101

AND-MASK:···0b0_0_0_1_0_00000000_000000000_00000_00000

RESULT:·····0b0_0_0_1_0_00000000_000000000_00000_00000
(often·check·non-zero,·or·check·equal·to·mask)
```

https://users.cs.cf.ac.uk/Dave.Marshall/C/drive.gif

Bit-Masks

- How do we set just the "Write Protection" bit to 0?

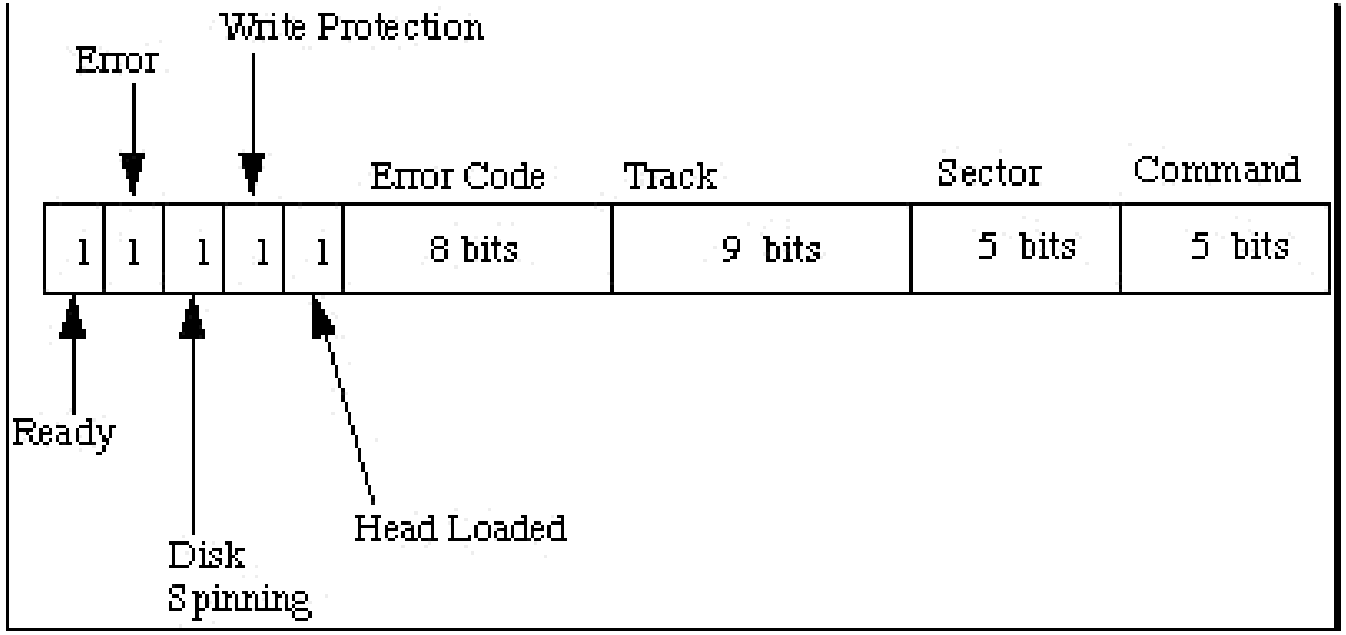



```
12  DATA:·····0b1_0_1_1_0_00011100_110000000_00101_01101
13
14  AND-MASK:···0b1_1_1_0_1_11111111_111111111_11111_11111
15
16  RESULT:·····0b1_0_1_0_0_00011100_110000000_00101_01101
17  (set·a·bit·to·a·0)
```

https://users.cs.cf.ac.uk/Dave.Marshall/C/drive.gif

Bit-Masks

- How do we set just the "Write Protection" bit to 1?

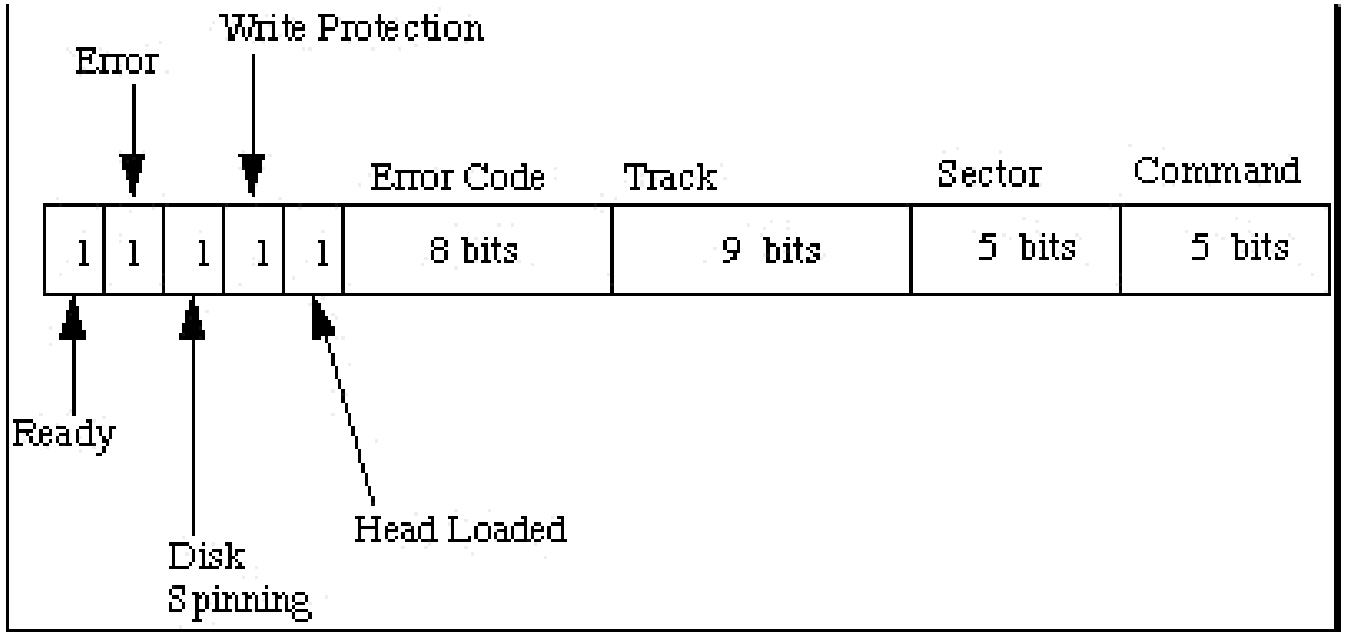



```
20  DATA:·····0b1_0_1_0_0_00011100_110000000_00101_01101
21
22  OR-MASK:···0b0_0_0_1_0_00000000_000000000_00000_00000
23
24  RESULT:····0b1_0_1_1_0_00011100_110000000_00101_01101
25  (set·a·bit·to·a·1)
```

https://users.cs.cf.ac.uk/Dave.Marshall/C/drive.gif

Bit-Masks

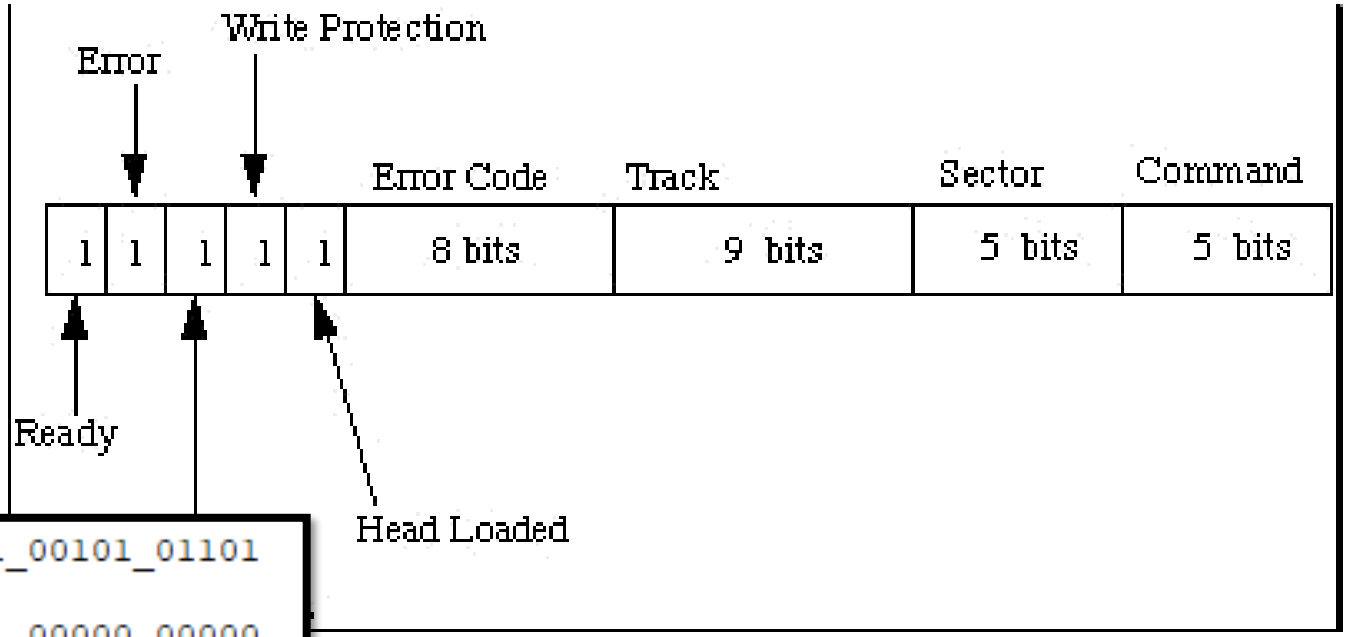
- How might we fill a register with just the “track” bits?
- Think for a while…



https://users.cs.cf.ac.uk/Dave.Marshall/C/drive.gif

Bit-Masks

- How might we fill a register with just the “track” bits?
- Think for a while…




```
28  DATA:·····0b1_0_1_1_0_00011100_101010101_00101_01101
29
30  AND-MASK:···0b0_0_0_0_0_00000000_111111111_00000_00000
31
32  RESULT:·····0b0_0_0_0_0_00000000_101010101_00000_00000
33
34  LOGICAL-RIGHT-SHIFT-10-BITS
35
36  RESULT:·····0b0_0_0_0_0_00000000_000000000_10101_01010
```

```
39    movi·r16,·0xB0E554AD····/*·0xB0E554AD·=·DATA··*/
40    movi·r17,·0x7FC00·····/*·0x7FC00·=·AND-MASK·*/
41    and·r16,·r16,·r17·····/*·apply·mask·*/
42    srli·r16,·r16,·10·····/*·shift-right·10·bits·*/
43    /*·r16·now·holds·"track"·bits·*/
44
```

https://users.cs.cf.ac.uk/Dave.Marshall/C/drive.gif

Bit-Masks

- Note .equ constants for readability & maintainability

```
39    movi r16, 0xB0E554AD    /* 0xB0E554AD = DATA  */
40    movi r17, 0x7FC00       /* 0x7FC00 = AND-MASK */
41    and r16, r16, r17       /* apply mask */
42    srli r16, r16, 10       /* shift-right 10 bits */
43    /* r16 now holds "track" bits */
44
```

```
37
38    .equ TrackBitsMask, 0x7FC00          /* 0b0...01111111110000000000 */
39    .equ TrackBitsStartingBit, 10        /* first track bit is bit 10  */
40    movi r16, 0xB0E554AD                 /* 0xB0E554AD = DATA  */
41    movi r17, TrackBitsMask              /* TrackBits = AND-MASK */
42    and r16, r16, r17                    /* apply TrackBits mask */
43    srli r16, r16, TrackBitsStartingBit  /* shift-right to right-edge */
44    /* r16 now holds "track" bits */
```

Register Bit-Masks

- More assembly and masking examples in NIOS IO slide deck!

Types of I/O Systems

- I/O Devices
  - LEDs
  - Keyboards
  - Switches
  - Displays
  - …

Types of I/O Systems

- I/O Devices
  - LEDs
  - Keyboards
  - Switches
  - Displays
  - …
- Receive data from input devices
- Send data to output devices

Types of I/O Systems

- I/O Devices
  - LEDs
  - Keyboards
  - Switches
  - Displays
  - …
- Receive data from input devices - like read data from memory
- Send data to output devices - like write data to memory

Types of I/O Systems

- Options:
  - Separate Memory & I/O Address Spaces (Intel)
  - Unified Address Spaces (Motorola, NIOS)
- Unified Address Spaces
  - Inputs performed with ldwio, a non-caching ldw
  - Outputs performed with stwio, a non-caching stw

Types of I/O Systems

- Options:
  - Separate Memory & I/O Address Spaces (Intel)
  - Unified Address Spaces (Motorola, NIOS)
- Unified Address Spaces
  - Inputs performed with ldwio, a non-caching ldw
  - Outputs performed with stwio, a non-caching stw
  - Addresses not used for memory may be used for I/O devices
  - Each device is allocated a certain range of addresses
  - Bit-Fields within that range of addresses are defined to have certain meaning to the device

PIO & Timer Devices

- PIO Core Devices
  - General Purpose I/O
  - Nice for “banks” of 2-state (1-bit, on/off) devices, ex: LEDs & Switches

PIO & Timer Devices

- PIO Core Devices
  - General Purpose I/O
  - Nice for “banks” of 2-state (1-bit, on/off) devices, ex: LEDs & Switches
- Timer Device
  - Commonly needed / application-independent device
  - Has both input (configuration) and output (timing state)

PIO & Timer Devices

- PIO Core Devices
  - General Purpose I/O
  - Nice for “banks” of 2-state (1-bit, on/off) devices, ex: LEDs & Switches
- Timer Device
  - Commonly needed / application-independent device
  - Has both input (configuration) and output (timing state)
- Note:
  - Addresses are byte-addressed
  - Offsets are by-word (4-bytes)
  - Effective Address = Port_Base_Address + 4*offset

PIO Core (Peripheral I/O device/port)

Table 9–2. Register Map for the PIO Core

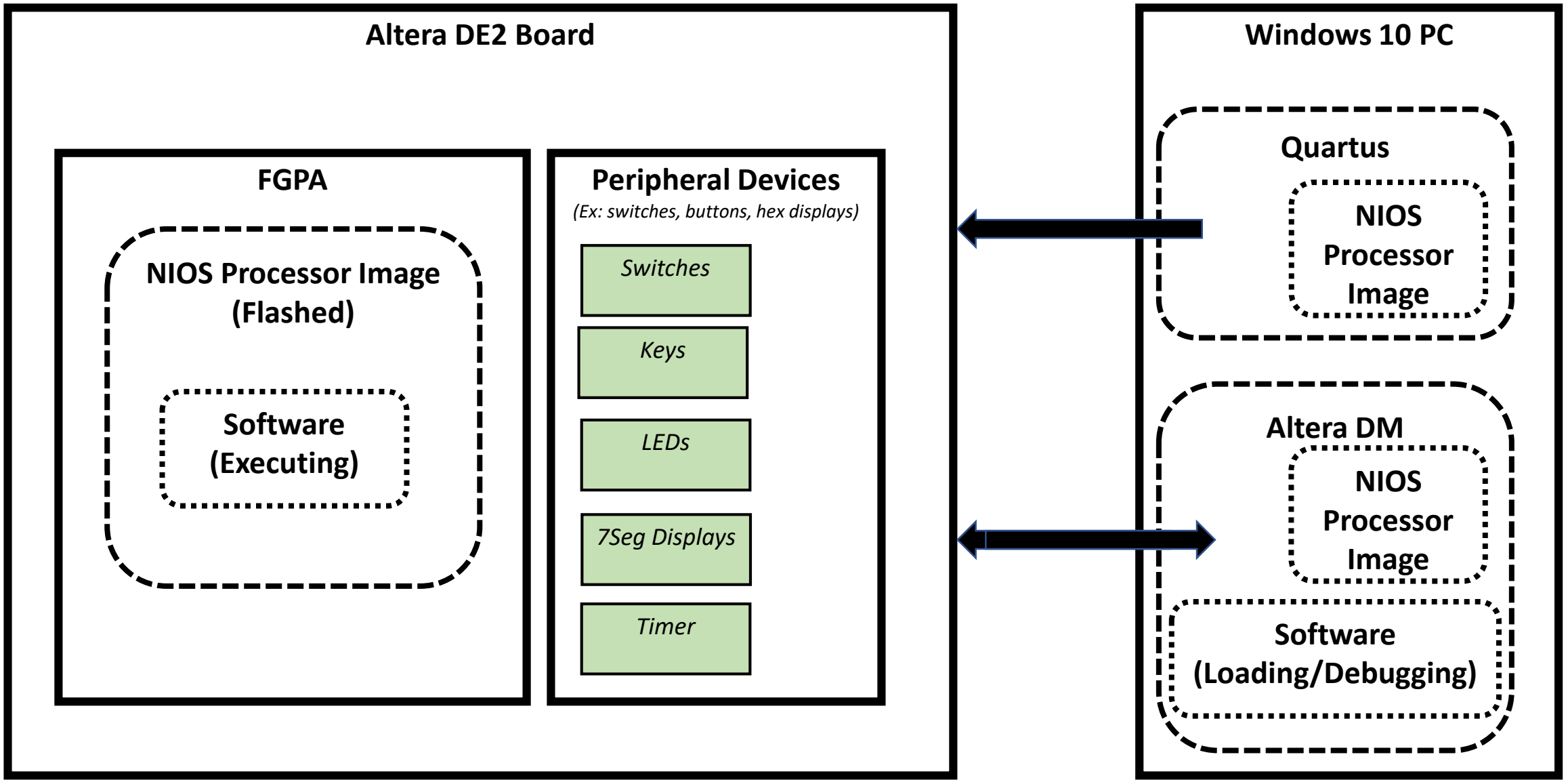
Offset	Register Name		R/W	(n-1)	...	2	1	0
0	`data`	read access	R	Data value currently on PIO inputs				
		write access	W	New value to drive on PIO outputs				
1	`direction` *(1)*		R/W	Individual direction control for each I/O port. A value of 0 sets the direction to input; 1 sets the direction to output.				
2	`interruptmask` *(1)*		R/W	IRQ enable/disable for each input port. Setting a bit to 1 enables interrupts for the corresponding port.				
3	`edgecapture` *(1), (2)*		R/W	Edge detection for each input port.				

Interval Timer Core

Table 24–3. Register Map—32-bit Timer

Offset	Name	R/W	Description of Bits						
			15	...	4	3	2	1	0
0	status	RW	*(1)*					RUN	TO
1	control	RW	*(1)*			STOP	START	CONT	ITO
2	periodl	RW	Timeout Period – 1 (bits [15:0])						
3	periodh	RW	Timeout Period – 1 (bits [31:16])						
4	snapl	RW	Counter Snapshot (bits [15:0])						
5	snaph	RW	Counter Snapshot (bits [31:16])						

Lab Components



PIO & Timer Devices

- Recommended Approach:
  1. Define .equ constants for all I/O device base addresses
  2. Define .equ constants for offsets (in bytes) to each I/O register
  3. Load the current device's base address constant into a register
  4. Use stwio and ldwio with the register and the offset constant

```
47      .equ TIMER_BASE, 0x10800
48      .equ PERIODH, 0xC
49      movia r16, TIMER_BASE
50      stwio r8, PERIODH(r16)
```

Lab SW05 Tips

- Don’t forget to initialize IO devices before use (if needed)
- Use .equ constants instead of hardcoding immediate values
- The .equ constants can be placed anywhere that an immediate value can be, including stwio offsets
- Immediate (.equ) values can be written in decimal, hex (0x) or binary (0b)
- Refer to “NIOS IO” slide deck for details on masking & IO devices

Getting Started

1. Setup project directory for SW05 on the **H: drive**
2. Download SW05 files from Moodle or S: drive
3. Note that there are new reference files for this lab, the ones from previous labs also still apply
4. Start following directions in the lab assignment document