This section will cover the core initialization process.

System Boot-up

- **Three major steps to Boot-up**
  - CPU initialization
    - Initialize the CPU to a known state. This includes GPR registers, CP0 registers, Cache, TLB, interrupt control and stack. More in next slides.
  - Minimum Device initialization for Loading OS
    - Whatever is needed to be able to load OS or main run loop.
  - OS-specific initialization.

MIPS

There are three steps to initializing a system, CPU, Boot device and OS initialization

+ CPU initialization initializes the CPU to a known state. That is what we are going to cover in this section of the course.

+ Next you would normally go on to initialize devices need to bring in the operating system.

+then you would start the OS and the OS would then initialize itself.

These last two step will not be covered here. You should consult your OS manual for more information.

Example of CPU Boot Code

- **Initialization of General Purpose Registers to known value**
  - Use the 'or' instruction to zero all of the registers.
    - Since register 0 is always 0, doing a logical or with itself and storing the results in a register will clear that register.

```
or    $1,$0, $0
or    $2,$0, $0
or    $3,$0, $0
or    $4,$0, $0
……
or    $31,$0, $0
```

MIPS

3

The first thing I recommend doing is to initialize the general purpose registers. This is not strictly necessary the CPU will function fine without it but it is a good software practice to do it.

+ to initialize the registers the code will move a zero using the ever constant zero register into the remaining registers. It does this by using the "OR" instruction. You can also write this code using the "move" instruction because the move instruction is really an alias for the or instruction.

MIPS32 Status Register - CP0 $12

- Status Register setting (0x0000ff02)

Status Register																		
31	27	26	25	24		22	21	20	19	18	17		15 8	7 5	4 3	2	1	0
CU	RP	FR	RE	MX	R	BEV	TS	SR	NMI	0	CEE	R	IM(7:0)	0	KSU	ERL	EXL	IE

CU	Co-processor Usability
RP	Reduced Power mode
FR	Indicates register mode 64-bit floating point unit
RE	Reverse Endian for User mode (Kernel references are not effected by this bit)
BEV	Bootstrap Exception Vector
TS	TLB Shutdown (multiple matching entry error)
SR	Indicates Soft Reset if set
NMI	Indicates NMI if set
IM	Interrupt Mask (sw, hw, timer) / IM(7:2) is IPL in Release 2 with EIC enabled
KSU	Operating Mode
ERL	Error Level (set for Reset, Soft Reset, NMI or Cache Error)
EXL	Exception Level (set for exception other than the ERL conditions)
IE	Global Interrupt Enable

MIPS

4

The Co processor zero status register controls the operating modes of the processor. The status register plays an important part in the initialization process so I will go into it in detail here

+ Bits 28 through 31 are the Coprocessor Usable bits. These bits control access to co processors zero through 3. Bit 28 controls access to coprocessor 0 when the cpu is in user mode. Bit 29 controls coprocessor 1 which is usually the floating point processor, bit 30 is reserved for the implementers that want to add their own special processor. You want to disable all these bits during the boot process and have the OS enable them if it will be using them.

+ The Reduce Power bit number 27 can be used to send a signal to an external clock that tells it that it can slow down to reduce power. This is usually used in conjunction with the wait instruction when the CPU is in a idle state. On boot the CPU should be at full power so this bit should be cleared.

+ The Floating Point Bit controls the number of FPU registers that are available it is used for compatibility with a MIPS one ISA. When this bit is cleared the CPU Floating point unit is in compatibility mode. It should be cleared and later controlled as the OS sees fit.

*+ The reverse Endian bit number 25 reverses the endianness of instructions fetched when set. This should be cleared and left

up to the OS to switch. As a side note there is no know OS that sets this bit.

+ The MX bit number 24 enables DSP instructions. This should be cleared and left to the OS to set if required.

+ The Boot Exception Vector bit 22 controls which set of exception vectors are used. Vectors were covered in the exception section of this course so I won't go over them here. This should be cleared so that the use normal exception vectors will be used.

+ the TLB Shutdown bit is set by the processor when it detects an attempt to create a duplicate TLB entry It will also generate a machine check exception. The exception routine should correct the condition and can clear the bit. This bit should be cleared to initialized it.

+ The Soft reset bit number 20 would indicate a soft reset condition. However in our cores it is hardwired to 0 because soft reset is not supported.

+ the NMI bit number 19 is a read only bit that is set when the CPU get a non maskable interrupt

+ The Core extend enable bit number 17 enables the use for the core extend instructions

*+ Bits eight through fifteen are the interrupt mask bits. View the exception section of the course for more information. For now these should be set to mask all interrupts and left to the OS to clear as device drivers are initialized.

+ The Kernel User Supervisor bits three and four should be cleared which puts the CPU into kernel mode

+ The error exception mode "ERL" bit two is set by the CPU at power up, NMI or Cache exception. It should be Cleared

+ The Regular Exception mode "EXL" bit 1 should be set to keep us in Kernel mode.

+ and last the Interrupt enable bit, bit zero should be cleared to disable interrupts and left to the OS to set when it has initialized its drivers and is ready to accept interrupts.

Example of CPU Boot Code

- **Assemble Code to Set Status register**

```
li      $11, 0x0000ff02
mtc0    $11, $12
```

MIPS

To set the status register as just described load the immediate value of FF 02 into a general purpose register and use the move to coprocessor zero instruction to move the registers value to the status register.

Example CPU Boot Code

- **Watch Lo register setting CP0 $18**
  - Disable Watch Exceptions by clearing the Watch Lo

```
mtc0   $0, $18
mtc0   $0, $18, 1
mtc0   $0, $18, 2
mtc0   $0, $18, 3
```

- **Watch Hi register setting CP0 $19**
  - Clear Watch Status Bits 0, 1 and 2 of the Watch Hi

```
li     $11, 0x7
mtc0   $11, $19
mtc0   $11, $19, 1
mtc0   $11, $19, 2
mtc0   $11, $19, 3
```

MIPS

6

The boot code needs to disable and clear watch point exceptions. There are four pairs of watch point registers.

+ The watch Lo registers are Co processor zero register eighteen select zero through three. Writing a zero to the four Watch Lo registers will disable all watch points.

+ The watch Hi registers are Co processor zero register nineteen select zero through three. Writing a seven to the four watch Hi registers will clear all watch conditions.

Example CPU Boot Code

- **Cause Register setting CP0 $13**
  - Code:
    - mtc0 $0, $13

MIPS

The Cause Register describes the cause of the most recent exception, enables vectored interrupts and enables the count register.

+ The boot code should clear the Cause register so there will not be spurious interrupts when interrupts are enabled.

Example CPU Boot Code

- **Config register setting CP0 $16**
  - Set Kseg 0 to cacheable, Write-Back and Write-Allocate (bits 0 and 1)

```
mfc0   $10, $16
or     $10, 0x3
mtc0   $10, $16
```

MIPS

The configuration register controls among other things the Cacheability of KSEG 0.

+ Setting bits zero and one set KSEG 0 to cacheable write back write allocate mode.

Example CPU Boot Code

- **Count register Clear CP0 #9**

```
mtc0    $0, $9
```

- **Compare register setting CP0 #11**
  - Set to delay first timer interrupt
  - Writing to Compare register clears the timer interrupt

```
li      $10, -1
mtc0    $10, $11
```

MIPS

9

The Compare register acts in conjunction with the Count register to implement a timer and timer interrupt function.

The timer interrupt is an output of the core. The Compare register maintains a stable value and does not change on its

own. The count and compare registers need to be initialized so the OS may later use them for timer interrupts.

+ Clear the count register by setting the counter to 0.

+ Setting the Compare register to the highest number possible will give us time to do the remaining steps in initializing the CPU. It also clears the timer interrupt which in this case is redundant because all interrupts have been cleared and disabled when we cleared the status and cause registers.

Example CPU Boot Code

- **Initializing Cache and TLB**
  - You must use only uncached regions until the cache is initialized.
  - You must use only unmapped regions until the TLB is initialized and you're ready to maintain it.
- **Cache initialization**
  - See Cache section
- **TLB initialization**
  - See TLB section

Now we are at the set to initialize the caches and the TLB.

Until they are initialized:

+ You can only use uncached and unmapped memory regions like Kseg1. Remember the boot vector is in KSEG1.

+ The initialization of Caches and the TLB is covered in the Cache and TLB sections of this course.

Example of CPU Boot Code

- **Clear Shadow set registers**
  - Read number of shadow sets
    - SRSCtl Register (CP0 12, 2) bits

```
mfc0    $11, $12, 2
ext     $12, $11, 26, 4
```

    - Note the use of ext, extract bit field, is a MIPS32 release 2 instruction and only used after we determined we had a release 2 processor

MIPS

Just like the general purpose register set we also want to initialize the shadow register sets to zero if we have them.

+ First the code will read the Shadow register set control register.

+ Then it will extract the Highest Shadow Set number which is the number of shadow register sets. These are bits twenty six through twenty nine so the extract instruction will start extracting from register eleven at bit twenty six and go for four bits and place the value in the low order bits of register twelve.

Example of CPU Boot Code

- **Clear Shadow set registers**
  - Loop to clear each set
    - At start check to see if there are any more register sets to clear. (If zero on first try, we don't have any SRs)

```
1:    beqz   $12, return
```

    - Set PSS to shadow set to be initialized
    - Note ins (Insert Bit field) is a release2 instruction being used to insert the Pervious SR set number

```
ins    $11, $12, 6, 4
mtc0   $11, $12, 2
```


Now the code sets up a loop.

+ First check to see if there are any shadow register sets to clear.

+ If there are, use the number in register twelve to set the shadow set we will be clearing. The code does this by using register twelve which holds the number of the highest shadow register set yet to be cleared, and inserting them into bits six through nine, the previous shadow register set field, of the Shadow register set control register that was read into register 12 in a previous step.

Then the move to coprocessor zero instruction moves the value in register eleven into the actual Shadow Register Set Control register.

Example of CPU Boot Code

- **Clear Shadow set registers**
  - Loop to clear each set
    - Use the wrpgpr (Write to GPR in previous register set)

```
wrpgpr  $1, $0
wrpgpr  $2, $0
wrpgpr  $3, $0
.
.
wrpgpr  $31, $0
```

    - Branch to top of loop

```
b       1b
add     $12, -1
```

MIPS

Setting the “previous shadow register set” causes all writes using the “write previous general purpose register” to be written to that shadow set. The code zeros out each register in the set in turn.

+ after that the code branches to the beginning of the loop and decrements the “highest register set counter” held in register twelve .

Example of CPU Boot Code

- **Set start address and jump to it**
  - Load the address of the code into EPC (exception program Counter) CP0 14

    ```
    li     $11, 0x80030120
    mtc0   $11, $14
    ```

  - Clear hazard

    ```
    ehb
    ```

  - Return from restart exception

    ```
    eret
    ```

MIPS

14

The code has done all initializations for the CPU and is ready to execute the next boot step.

+ The code loads the starting address of the next part of the code into the Error Program Counter. Note the address in the example is in the KSEG 0 memory section so it is Cachable and once jumped to the code will be running out of cache. The actual address will be determined by your code but it should still be in the KSEG 0 memory region.

+ the code executes a ehb instruction to clear any hazard barrier

+ and then the return from exception instruction to exit exception mode and jump to the address held in the error program counter.