Project

Muhammad Tahir*

Lecture 12

*Professor, Electrical Engineering Department, University of Engineering and Technology Lahore

Contents

Project Description

- RV32I with machine mode only
- Harvard bus architecture
- Data hazards and their resolution
- User-level ISA Spec. 2.2 to be followed
- Privileged architecture Spec. 1.11

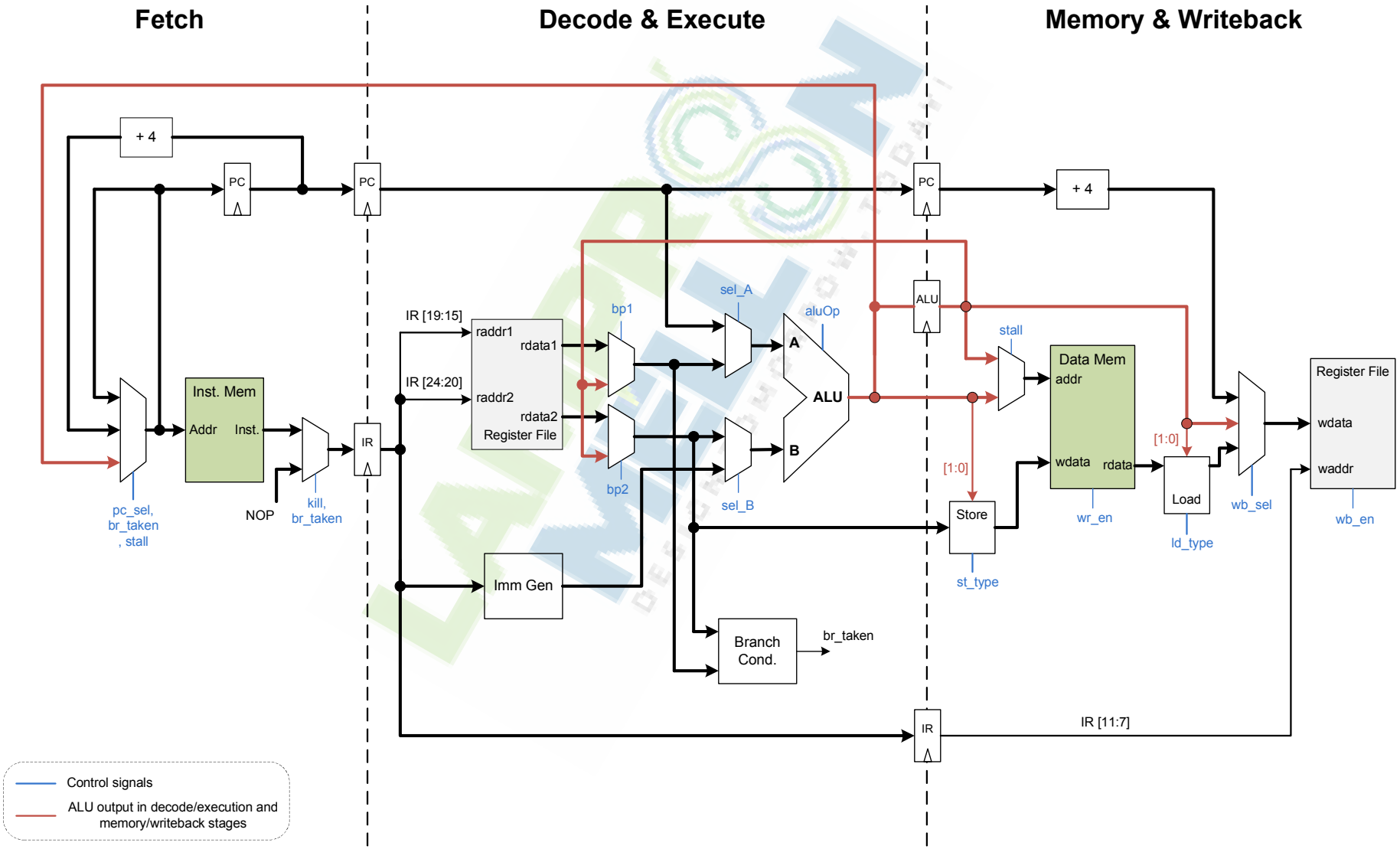
Processor Core Modules

- ALU
- Conditional Branch
- Immediate value generation module
- Register file
- Data memory and interface module
- Instruction memory and interface module
- The controller
- The datapath
- CSR

Microarchitecture

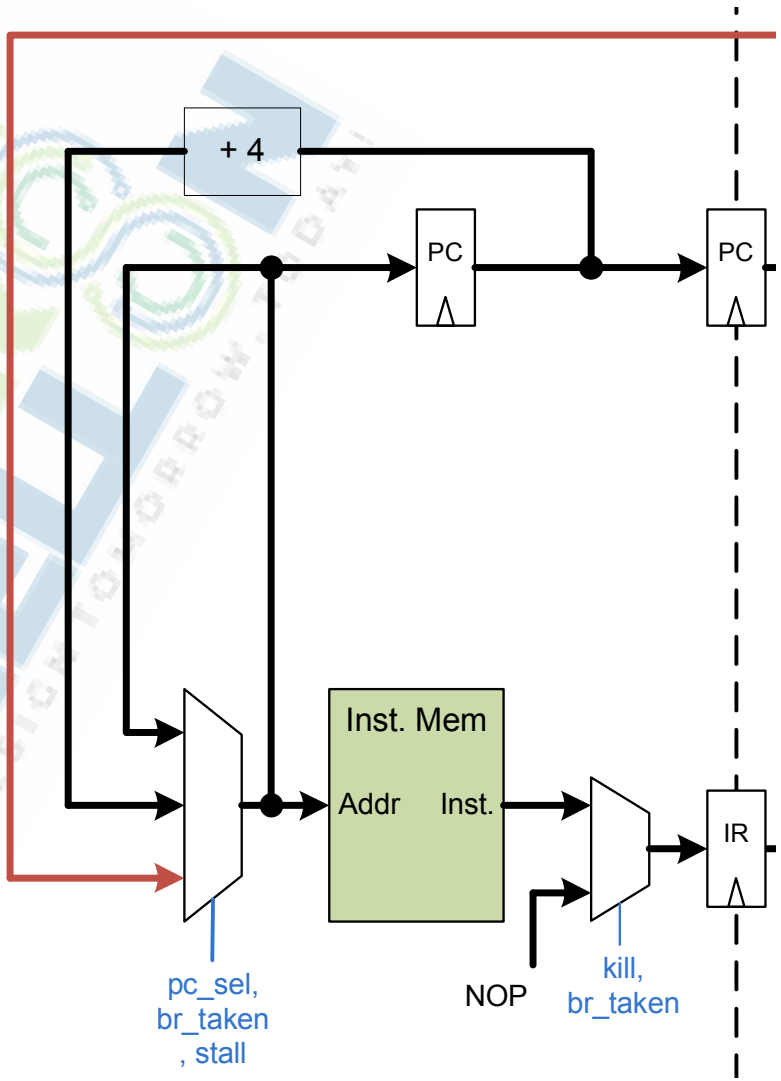
- Single core, 3-stage pipeline
- Pipeline stages are:
  - Instruction Fetch
  - Decode & Execute
  - Memory & Writeback
- No branch prediction

Microarchitecture



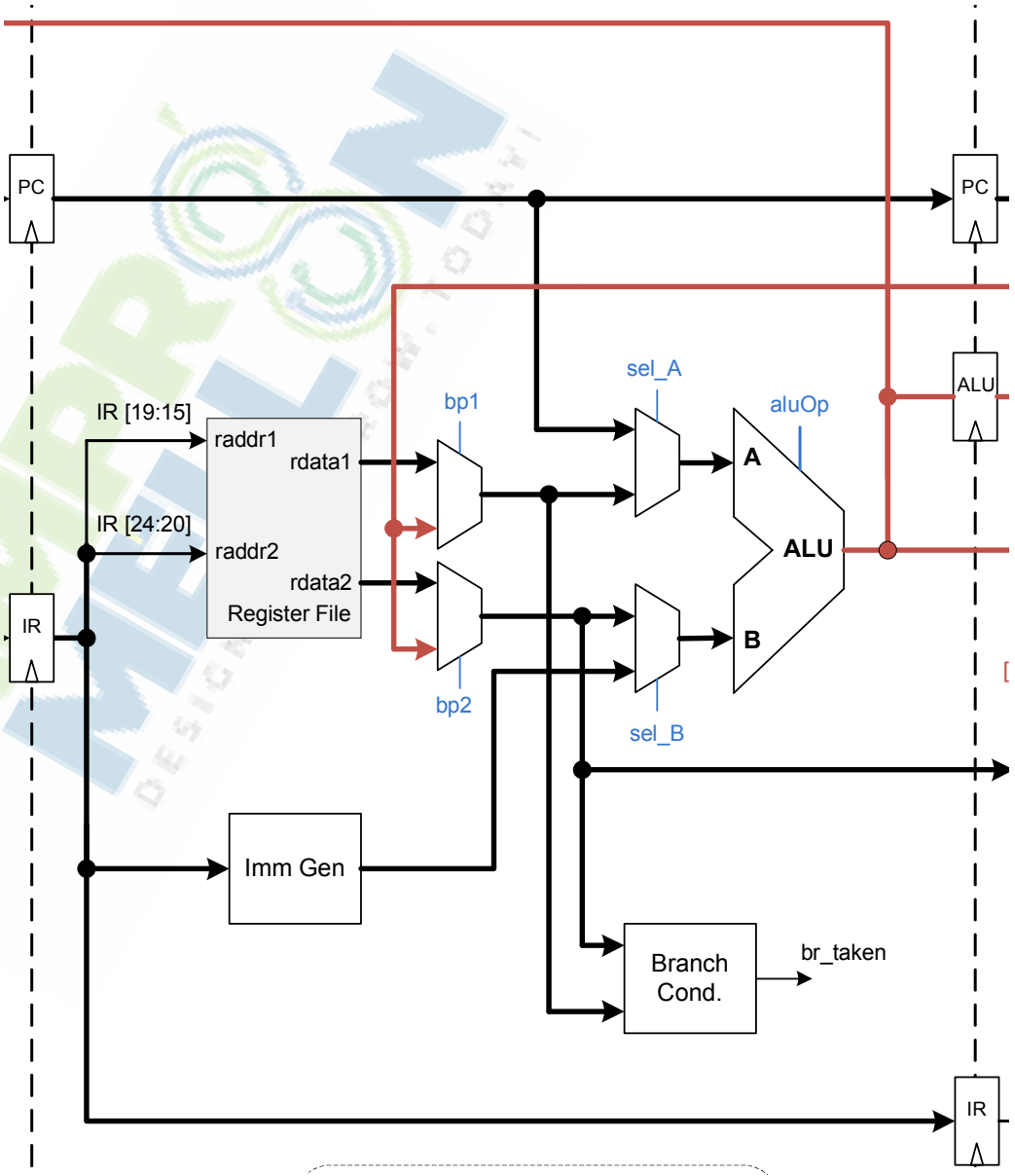
Microarchitecture Cont'd

- Fetch stage
- Uses synchronous memory
- No branch prediction



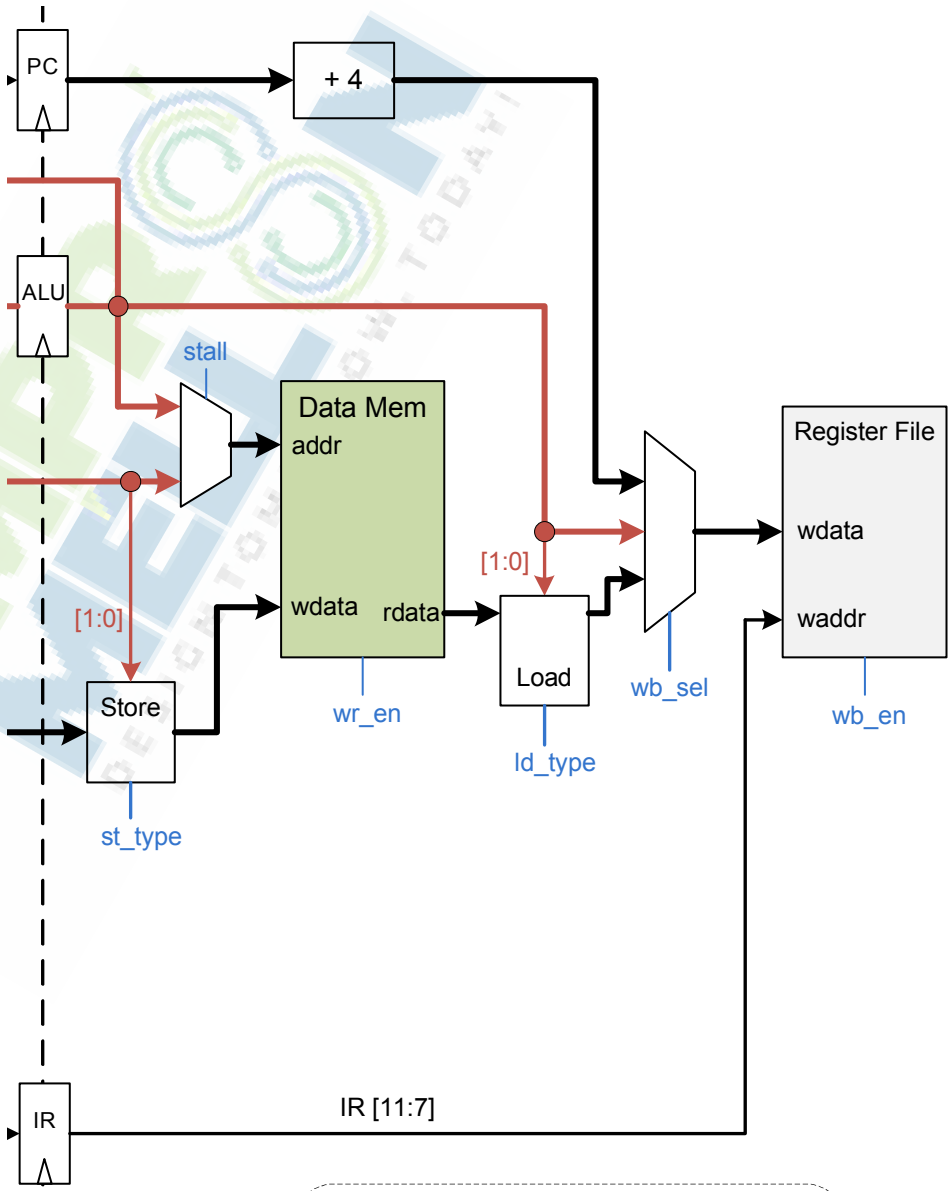
Microarchitecture Cont'd

- Decode & execute stage
- With data forwarding

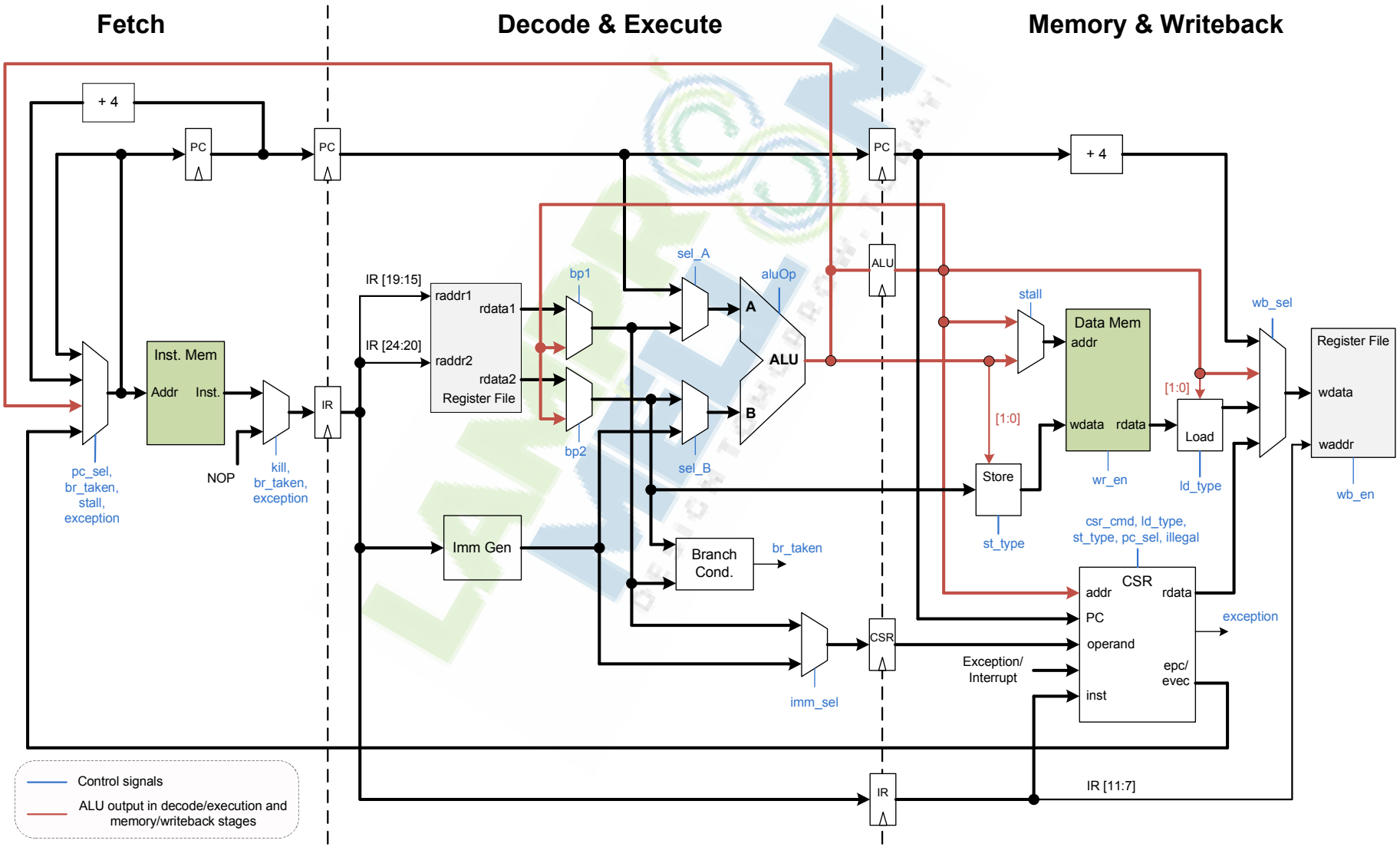


Microarchitecture Cont'd

- Memory and writeback stage
- Uses synchronous memory for data
- Missing CSRs implementation



Microarchitecture Cont'd



RISC V Privilege Levels

- Privilege levels are used to provide protection between different components of the software stack
- At a given time, a RISC-V hardware thread (hart) is running at some privilege level
- Privilege level is encoded as a mode in one or more CSRs (control status registers)

Level	Encoding	Name	Abbreviation
0	00	User/application level	U
1	01	Supervisor level	S
2	10	Reserved	
3	11	Machine level	M

Table: RISC V privilege levels.

RISC V Privilege Levels Cont'd

- Supported modes at different privilege levels

No. of Level	Supported Modes	Usage
1	M	For simple embedded systems
2	M, U	For secure embedded systems
3	M, S, U	For systems running Linux type operating systems

Table: Combinations of privilege levels that can be supported.

Machine Mode Registers

Table: RISC-V machine-level CSR groups and addressing.

(csr[11:10])	(csr[9:8])	Hex Address	Description
00	11	0x30x	Machine trap/interrupt setup
00	11	0x32x, 0x33x	Machine counter setup
00	11	0x34x	Machine trap handling
00	11	0x3Ax, 0x3Bx	Machine memory protection
01	11	0x7Ax	Debug/Trace registers (shared with Debug mode)
01	11	0x7Bx	Debug mode registers
10	11	0xBxx	Machine counter/timers
11	11	0xF1x	Machine information registers

Machine Mode Registers Cont'd

Table: Machine trap/interrupt setup registers.

Access Privilege	Address	Register Name	Description
RW	0x300	mstatus	Machine status register.
RW	0x301	misa	ISA and extensions.
RW	0x302	medeleg	Machine exception delegation register.
RW	0x303	mideleg	Machine interrupt delegation register.
RW	0x304	mie	Machine interrupt-enable register.
RW	0x305	mtvec	Machine trap-handler base address.
RW	0x306	mcounteren	Machine counter enable.

Machine Mode Registers Cont'd

Table: Machine trap handling registers.

Access Privilege	Address	Register Name	Description
RW	0x340	mscratch	Scratch register for machine trap handlers.
RW	0x341	mepc	Machine exception program counter.
RW	0x342	mcause	Machine trap cause.
RW	0x343	mtval	Machine bad address or instruction.
RW	0x344	mip	Machine interrupt pending.

An Example Program

```
int add(int x, int y) {
 return x+y;
}

int main(void) {
    // declare some variables
    int  x = 123, y = 987, z = 0;

    // call the user function
    z = add(x,y);

    // endless loop
    while(1){}
}
```

Example Startup File

```
.equ CSR_MSTATUS, 0x300
.equ MSTATUS_MIE, 0x00000008
.equ CSR_MTVEC,   0x305

 # Main interrupt vector table entries
.global vtable
.type vtable, %object
.section .text.vector_table,"a",%progbits

# this entry is to align reset_handler at address 0x04
  .word    0x00000013
  j        reset_handler
  .align   2
vtable:
  j        default_interrupt_handler
  .word    0
  .word    0
  j        msip_handler
  .word    0
  .word    0
```

Example Startup File Cont'd

```
    .word     0
    j         mtip_handler
    .word     0
    .word     0
    .word     0
    .word     0
    .word     0
    .word     0
    .word     0
    .word     0
    j         user_handler
    .word     0
    .word     0
```

Example Interrupt Service Routine

```
# RISC-V Interrupt Service Routines (ISRs)
# ALL supported ISRs should be put here

.section .text.isr

# User interrupt handler
.globl user_handler
user_handler:
  nop
  # you can call user ISR here and then return using 'mret'
  mret
```

Building the Example Program

- The build process steps

```
riscv64-unknown-elf-gcc -c -o build/main.o src/main.c -march=rv32i -mabi=ilp32

riscv64-unknown-elf-as -c -o build/startup.o src/startup.s -march=rv32i -mabi=
    ilp32

riscv64-unknown-elf-as -c -o build/isr.o src/isr.s -march=rv32i -mabi=ilp32

riscv64-unknown-elf-gcc -o build/program.elf build/startup.o build/isr.o build/
    main.o -T linker.ld -nostdlib -march=rv32i -mabi=ilp32

riscv64-unknown-elf-objcopy -O binary --only-section=.data* --only-section=.text
    * build/program.elf build/main.bin

hexdump build/main.bin > build/main.hex

python maketxt.py build/main.bin > build/main.txt

riscv64-unknown-elf-objdump -S -s build/program.elf > build/program.dump
```

- The output file `main.txt` is loaded to the instruction memory

Reading List I

- Read User Manual for the instruction set and its architecture [Waterman et al., 2016b].
- For Control and Status register description consult Privileged architecture [Waterman et al., 2016a].

References

Waterman, A., Lee, Y., Avizienis, R., Patterson, D. A., and Asanovic, K. (2016a).

The risc-v instruction set manual volume ii: Privileged architecture version 1.9.

EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-129.

Waterman, A., Lee, Y., Patterson, D. A., and Asanović, K. (2016b).

The risc-v instruction set manual, volume i: User-level isa, version 2.1.

EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-129.