2015-16 Project

Minimum requirements

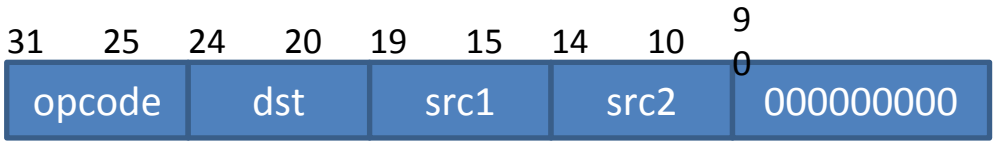
1. Basic Instruction Set

- Architectural state
  - 32 registers, each 32b (r0 through r31)
  - Special register rm0, 32b, for holding the PC the OS should return to on exceptions
  - Special register rm1, 32b, for holding an @ for certain exceptions
  - [optional] Special register rm2, 32b, holding information on the type of exception
- Interface to OS
  - Processor boots from address 0x1000
  - Exceptions will always jump to address 0x2000

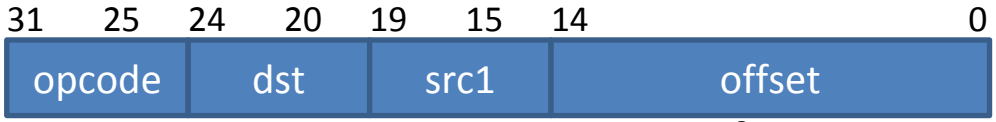
- Basic Instruction Set
  - Each instruction is 32bits.
  - ADD r1, r2 → r3 // Add two registers
  - SUB r1, r2 → r3 // Subtract two registers
  - MUL r1, r2→r3 // Multiply two registers
  - LDB 80(r1) → r0 // Load Byte; base register + offset
  - LDW 80(r1) → r0 // Load Word; base register + offset
  - STB r0 → 80(r1) // Store Byte; base register + offset
  - STW r0 → 80(r1) // Store Word; base register + offset
  - BEQ r1, offset // if r1==r2, PC=PC+offset
  - JUMP r1, offset // PC = r1 + offset

Suggested Encoding

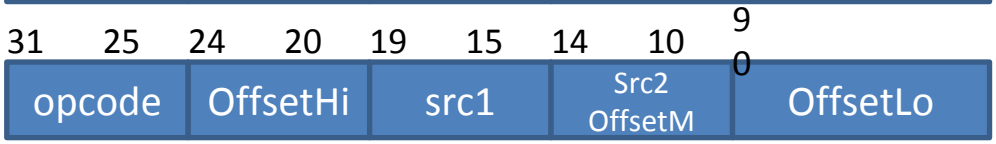
R-type instructions



M-type instructions



B-type instructions



- R-type: ADD, SUB, MUL
- M-type: LDB,LDW,STB,STW
- B-type:
  - BEQ: Offset[14:0] = {OffsetHi, OffsetLo}
  - JUMP: Offset[19:0] = {OffsetHi, OffsetM, OffsetLo}
- Optional instructions
  - MOV rm0, dst: encoded as M-type, src1=0, offset=0
  - TLBWRITE r0,r1: encoded as B-type: src1=r0, src2=r1, offsetLo=0 for ITLB, offsetLo=1 for DTLB
  - IRET: encoded as B-type: src1=0, src2=0, offsethi=0, offsetlo=0

Instruction	Opcode
ADD	0x0
SUB	0x1
MUL	0x2
LDB	0x10
LDW	0x11
STB	0x12
STW	0x13
MOV	0x14
BEQ	0x30
JUMP	0x31
TLBWRITE	0x32
IRET	0x33

2. Caches

- Instruction Cache
  - 4 cache lines, 128b per cache line
  - You can pick any replacement policy you want
  - You can pick any associativity you want
- Data Cache
  - 4 cache lines, 128b per cache line
  - You can pick any replacement policy you want
  - You can pick any associativity you want
- Memory
  - 5 cycles to go to memory
  - 5 cycles to return data from memory to processor
- [optional] Support unaligned LDW, STW

3. PIPELINE BASICS

- 5 or more stage pipeline
- Artificially make the MUL instruction take 5 execution stages.
  - So the minimal pipeline for it would be:
    - F, D, M1, M2, M3, M4, M5, WB
  - You can add other stages to the above pipeline if you need it in your machine
- Full set of bypasses
- Store buffer of some sort
- ROB or HF or FF

4. Adding Virtual Memory

- Virtual memory architecture
  - Virtual Address is 32bits
  - Physical Address is 20bits
  - Pages are 4KB
- Translation Scheme
  - Simply add some displacement to bits [31:12]
  - For example: PA = VA + 0x8000;
- Privilege Scheme
  - We add another special register rm4
  - Bit 0 in rm4 holds the current privilege of the machine
    - 0 for users
    - 1 for supervisor
- The machine boots in supervisor mode
- When in supervisor mode, virtual memory is disabled
  - This will simplify your mini-os and your logic
  - So don't look up the TLB when rm4[0] = 1
- Therefore:
  - Processor boots from PHYSICAL address 0x1000
  - Exceptions jump to PHYSICAL address 0x2000

4. Changes for Virtual Memory

- Add an iTLB and a dTLB
- On a TLB miss:
  - Save the faulting PC to rm0
  - Save faulting @ to rm1
  - [optional] save additional info to rm2 that your mini-os may need
  - Jump to physical address 0x2000

- The Mini-OS code will do something like
  - Store r5 → [A]
  - Store r6 → [B]
  - MOV rm1 → r5
  - ADD r5, #8000 → r6 // r6 represents the translation of r5
  - TLBWRITE r5, r6 // (r5 = virtual @, r6 = physical @)
  - Load [A] → r5
  - Load [B] → r6
  - IRET
- Notice we added 3 new instructions
  - MOV from rm1 to r5
  - TLBWrite r5, r6 = adds the pair <va,pa> to our tlb; faults if not privileged
  - IRET = jumps to the @ in rm1 and lowers the PSW to user

Performance Tests

- Note: you of course have to have other functional tests beyong these 3 performance tests
- Count how many cycles the three following tests take on your machine

- Buffer_sum

```
int a[128], sum = 0;
for (i=0; i<128; i++) { sum += a[i]; }
```

- Mem_copy

```
int a[128], b[128];
for (i=0; i<128; i++) { a[i] = 5; }
for (i=0; i<128; i++) { b[i] = a[i]; }
```

- Matrix multiply

```
int a[128][128], b[128][128], c[128][128];
for (i=0; i<128; i++) {
     for(j =0;j<128;j++) {
          c[i][j] = 0;
          for(k = 0; k <128; k++ ) {
               c[i][j] = c[i][j] + a[i][k] * b[k][j];
          }
     }
}
```

Bonus

- Branch
- DMA engine
- Out-of-order