Implemented Instructions:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instruction | Opcode / Opcode Extension | Example | Description | Flags |
| ADD | 0000 0101 | ADD *src*, *dest*  ADD r0, r1  r1 = r1 + r0 | Integer addition | Carry  Overflow |
| ADDI | 0101 xxxx | ADDI *imm*, *dest*  ADD $2, r1  r1 = r1 + 2 | Integer addition with sign-extended immediate | Carry  Overflow |
| ADDU | 0000 0110 | ADDU *src*, *dest*  ADD r0, r1  r1 = r1 + r0 | Unsigned integer addition | None  (CR16 PSR flags are not affected by ADDU instruction) |
| ADDUI | 0110 xxxx | ADDUI *imm*, *dest*  ADDUI $2, r0  r0 = r0 + 2 | Unsigned integer addition with zero-extended immediate | None |
| ADDC | 0000 0111 | ADDC *src, dest*  ADDC r0, r1  r0 = r1 + r0 + C | Integer addition with carry | Carry  Overflow |
| ADDCI | 0111 xxxx | ADDCI *imm*, *dest*  ADDCI $2, r1  r1 = r1 + 2 + C | Integer addition with sign-extended immediate and carry | Carry  Overflow |
| SUB | 0000 1001 | SUB *src*, *dest*  SUB r0, r1  r1 = r1 - r0 | Integer subtraction | Carry  Zero  Overflow  Low  Negative |
| SUBI | 1001 xxxx | SUBI *imm*, *dest*  SUBI $2, r0  r0 = r0 - 2 | Integer subtraction with sign-extended immediate | Carry  Zero  Overflow  Low  Negative |
| SUBC | 0000 1010 | SUBC *src*, *dest*  SUBC r0, r1  r1 = r1 – (r0 + C) | Integer subtraction with carry | Carry  Zero  Overflow  Low  Negative |
| SUBCI | 1010 xxxx | SUBCI *imm*, *dest*  SUBCI $2, r1  r1 = r1 – (2 + C) | Integer subtraction with sign-extended immediate and carry | Carry  Zero  Overflow  Low  Negative |
| CMP | 0000 1011 | CMP *src1*, *src2*  CMP r0, r1 | Compare Integer.  PSR.Z = 1 if src1=scr2  PSR.N = 1 if src1<scr2 (signed)  PSR.L = 1 if scr1<scr2 (unsigned) | Zero  Negative  Low |
| CMPI | 1011 xxxx | CMPI *imm*, *src2*  CMPI $0, r0 | Compare Integer.  PSR.Z = 1 if imm=scr2  PSR.N = 1 if imm<scr2 (signed)  PSR.L = 1 if imm<scr2 (unsigned) | Zero  Negative  Low |
| AND | 0000 0001 | AND *src*, *dest*  AND r0, r1  r1 = r1 & r0 | Bitwise Logical AND | None |
| ANDI | 0001 xxxx | ANDI *imm*, *dest*  ANDI 0x55, r1  r1 = r1 & 0x55 | Bitwise Logical AND with zero-extended immediate | None |
| OR | 0000 0010 | OR *src*, *dest*  OR r0, r1  r1 = r1 | r0 | Bitwise Logical OR | None |
| ORI | 0010 xxxx | ORI *imm*, *dest*  ORI 0x55, r1  r1 = r1 | 0x55 | Bitwise Logical OR with zero-extended immediate | None |
| XOR | 0000 0011 | XOR *src*, *dest*  XOR r0, r1  r1 = r1 ^ r0 | Bitwise Logical XOR | None |
| XORI | 0011 xxxx | XORI *imm*, dest  AND 0x55, r1  r1 = r1 ^ 0x55 | Bitwise Logical XOR with zero-extended immediate | None |
| MOV | 0000 1101 | MOV *src*, *dest*  MOV r0, r1  r1 = r0 | Move | None |
| MOVI | 1101 xxxx | MOV *imm*, *dest*  MOV $7, r0  r0 = $7 | Move with zero-extended immediate | None |
| LSH | 1000 0100 | LSH *count*, *dest*  LSH r0, r1  r1 = r1 << r0 (r0 > 0)  r1 = r1 >> r0 (r0 < 0) | Logical Shift Integer  If count is positive, left shift, if count is negative, right shift | None |
| LSHI | 1000 000s | LSHI *imm*, *dest*  LSHI $1, r1  r1 = r1 << 1  LSHI $-1, r1  r1 = r1 >> 1 | Logical Shift Integer Immediate  s->0 right shift  s->1 left shift | None |
| ASHU | 1000 0110 | ASHU *count*, *dest*  ASHU r0, r1  r1 = r1 <<< r0 (r0 > 0)  r1 = r1 >>> r0 (r0 < 0) | Arithmetic Shift | None |
| ASHUI | 1000 001s | ASHUI *imm*, *dest*  ASHUI $1, r1  r1 = r1 <<< 1  LSH $-1, r1  r1 = r1 >>> 1 | Arithmetic Shift with immediate  s->0 right shift  s-> left shift | None |
| LUI | 1111 xxxx | LUI *imm*, *dest*  LUI 0x55, r0  r0 = 0x5500 | Load upper immediate | None |

Postponed Instructions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instruction | Opcode / Opcode Extension | Example | Description | Flags |
| LOAD | 0100 0000 |  | Load | None |
| STOR | 0100 0100 |  | Store | None |
| Bcond | 1100 xxxx |  | Conditional Branch | None |
| Jcond | 0100 1100 |  | Conditional Jump | None |
| JAL | 0100 1000 |  | Jump and Link |  |

In the previous lab, I did not ask you to write a report. So you will write a combined report on the ALU design

and synthesis, along with regfile design and datapath integration. Submit a 7-8 page report that describes:

• A list of instructions/opcodes that have been implemented so far, and what has been postponed until the design

of the DECODE machine (e.g., some of you may have postponed “immediate operations” for later).

• regfile design (ports, mux/buf-based, the basic block diagram, control signals, etc.)

• overall organization, data-transfer between the ALU + Regfiles

• Synthesis stats - Area (number of LUTs/ALMs, FFs, BUFs, etc. used), Delay (longest paths, etc.)

Once this lab is successfully completed, we will address: i) memory organization, ii) Block-ram and S/Dram

options available on board, iii) and how they will be integrated to develop a fetch-decode framework; i.e., how to

design the CPU controls