Laboratory Work 5 Preliminary work

② 1.2 ISA Configuration

We have a field for each distinct criterion in the instruction.

Binary codes are assigned to the each instruction in the entire ISA.

Instructions are 16 bits. According to the datapath, conditional information is just needed for conditional branch instructions.

① Let's have a look at the overwiev

there four types of instructions. These are Data-processing, shift, branch and memory instructions

Data processing

op	cmd	rd	rn	rm	cond
15:14	13:11	10:8	7:5	4:2	1:0

op=00

cmd	
000 →	add rd,rn,rm
001 →	addi
010 →	sub rd,rn,rm
011 →	subi

cmd	
100 →	and rd,rn,rm
101 →	orr rd,rn,rm
110 →	xor rd,rn,rm
111 →	clr rd

cond	
00 →	Equivalent (EQ)
01 →	not equivalent (NE)
10 →	lower unsigned (LO)
11 →	higher or same unsigned (HS)

shift instructions

op	shift type	rd	rn		cond
15:14	13:11	10:8	7:5		1:0

shift rn & store at rd

op=01

Shift type (shift rn & store at rd)	
000 →	rol rd,rn
001 →	ror rd,rn
010 →	lsl rd,rn
011 →	asr rd,rn
100 →	lsr rd,rn

cond	
00 →	EQ
01 →	NE
10 →	LO
11 →	HS

Memory inst.

op	typeM	rd	rn	imm5
15:14	13:12	10:8	7:5	4:0

OP = 10

typeM

00 → ldr rd, [rn, imm5]

01 → ldi rd, #Inst7:0

10 → str rd, [rn, imm5]

Branch inst.

op	typeB		imm8
15:14	13:11		7:0

OP = 11

typeB		typeB	
000 → B		100 → BNE	
001 → BL		101 → BC	
011 → BEQ		110 → BNC	

OP = 11 typeB = 010 ⇒

op	typeB		Rm (3 bits)	
15:14	13:11		4:2	

⇒ BI

(*) indirect branch, the target address is specified indirectly either through memory or general purpose registers.

(*) BI takes Rm as its operand and causes a branch to address saved in Rm.

12 ISA Configuration

Mnemonic	Name		Operation
ADD	Addition	add rA,rB,rC	rA ← rB + rC
ADDI	Addition indirect	add rA, rB,[#address]	rA ← rB + MEM[address]
SUB	Subtraction	sub rA,rB,rC	rA ← rB − rC
SUBI	subtraction indirect	sub rA,rB,[#address]	rA ← rB − MEM[address]
AND	logical and	and rA,rB,rC	rA ← rB & rC
ORR	logical or	orr rA,rB,rC	rA ← rB | rC
XOR	logical xor	xor rA,rB,rC	rA ← rB ^ rC
CLR	clearregister	clr rA	rA ← 0
ROL	rotate left	rol rA	rA ← {rA[6:0], rA[7]}
ROR	rotate right	ror rA	rA ← {rA[0], rA[7:1]}
LSL	logical shift left	lsl rA	rA ← {rA[6:0], 0}
ASR	arith. shift right	asr rA	rA ← {rA[7], rA[7:1]}
LSR	log. shift right	lsr rA	rA ← {0, rA[7:1]}
LDR	load register frommem.	ldr rA, [#address]	rA ← MEM[#address]
LDI	load imm. to reg.	ldi rA, #data	rA ← #data
STR	store from reg to mem.	str rA, [#address]	MEM[#address] ← rA
B	branch uncond.	add PC, (PC+8), imm8	PC ← (PC+8) + imm8

BL	branch with link	sub LR, (PC+8), 4
		add PC, (PC+8), imm8
			LR ← (PC+8)−4;
			PC ← (PC+8) + imm8

BI	branch indirect	mov PC, [Rm]	PC ← [Rm]
BEQ	branch if zero	Z=1 then add PC,(PC+8), imm8	if Z=1, then PC ← PC+8 + imm8
BNE	branch if not zero	Z=0 then add PC,(PC+8), imm8	if Z=0, then PC ← PC+8, + imm8
BC	branch if carry set	C=1 then add PC,(PC+8), imm8	if C=1 then, PC ← PC+8 + imm8
BNC	branch if carry clear	C=0 then add PC,(PC+8), imm8	if C=0 then, PC ← PC+8 + imm8

① ordering of the representation important to understand more clearly

For example sub rA,rB,rC implies rA ← rB − rC
sub rA,rC,rB implies rA ← rC − rB } totaly different operations. Therefore, operands order is important.

(2) I squeezed all the needed conditional information related to the instruction in a minimum number of bits.

Therefore, in my design instructions are 16 bits.

If I used 32 bits instructions, memory utilization would increase.

To eliminate unnecessary memory utilization, I have used 16 bits instructions. However, control unit complexity increases. The trade-off between memory utilization and component consumption can be evaluated according to the system requirements. I try to decrease memory utilization. When I used 32 bits instructions, I would obtain a simpler control unit with reduced number of functional components

(1.2.1) Multi-Cycle Controller Unit Design

Fetch => 1 clock cycle

Decode => 1 clock cycle

Execute => 1, 2, 3 clock cycle it depens on the instruction type.

FSM is going to be designed.

(R) ~~(A)~~ END ISA => 1111_1111_1111_1111 => that forces the execution to be halted

RUN / RESET are necessary signals.

RUN: incite the execution of the code from current instruction.

RESET: terminates the operation, sets the PC to very first slot in memory.

1 Control signal for the fetch, decode, execute cycles

Fetch -PHASE S0

AdrSrc=00

PCWrite= 1

ALUSrcA=1

ALUSrcB=10

ALUControl=4'b0000

ResultSrc=10

IRWrite=1

RegWrite=0

Decode -PHASE S1

RegSrc(specified according to OP)

PCWrite=0

MemWrite=0

IRWrite=0

RegWrite=0

ALUSrcA=1

ALUSrcB=2'b10

ALUControl=0000 (add)

ResultSrc=2'b10

EXECUTE-PHASE

DATA PROCESSING & SHIFT DATA

S2 Execute_ALU

ALUSrcA=0

ALUSrcB=00

ALUControl (from 0 to 11)

RegSrc = 3'b100

ResultSrc= 2'b10

RegWrite=0

S3 ALU_WB

ResultSrc= 2'b00

RegWrite=1

MEMORY INSTRUCTIONS

S4 ALU_WB_LDI (cycle3 LDI)

ImmSrc=1;

ALUSrcB= 2'b01;

ResultSrc=2'b11;

RegWrite=1;

S5 memADR (cycle3 for STR & LDR)

ImmSrc=0

ALUSrcB=2'b01

ALUSrcA=0;

ALUControl=4'b0000

S6 memWRITE (cycle 4 for STR)

ResultSrc= 2'b00

AdrSrc= 2'b01

MemWrite=1

S7 MemRead (cycle 4 for LDR)

ResultSrc=2'b00

AdrSrc= 2'b01

MemWrite=0

S8 mem_WB (cycle 5 for LDR)

ResultSrc= 2'b01

RegWrite=1

BRANCH INSTRUCTIONS

S9 (cycle 3 for B, BEQ, BNE, BC, BNC)
Branch_EX

PCWrite=(according to condition)

ALUSrcA=0

ALUControl=4'b0000

ALUSrcB=2'b01

ResultSrc =2'b10

S10 BL-EX (cycle 3 for BL inst.)

PCWrite=1

ALUSrcA=0

ALUControl=4'b0000

ALUSrcB=2'b01

ResultSrc=2'b10

RegWrite=1

S11 BI_PCWrite (cycle 3 for BI)

ALUSrcB=2'b00

ResultSrc= 2'b11

PCWrite=1

- Fetch is completed in 1 clock cycle.
- Decode is completed in 1 clock cycle.
- According to the instruction execution clock cycle number changes from 1 to 3.