1.2 ISA Configuration

⊗ supports → general purpose computing → contain different instructions

sufficient variety of addressing modes (direct, indirect, immediate)

⊗ own mnemonics for the instructions and define the ordering of the operands

group the instructions that will use the same path for execution

(a) Data processing instructions

- => addition
- => addition indirect
- => subtraction
- => subtraction indirect
- => and
- => or
- => xor
- => clear

(b) Shift instructions

- => rotate left
- => rotate right
- => shift left
- => arithmetic shift right
- => logical shift right

(c) Branch instructions

- => Branch unconditional
- => Branch with link
- => Branch indirect
- => Branch if zero
- => Branch if not zero
- => Branch if carry set
- => Branch if carry clear

(d) memory instructions

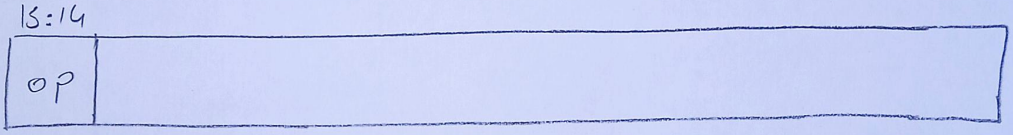
- => load to register from memory
- => load immediate to register
- => store from register to memory

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	clear register	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 from mem.	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

* ordering of the representation important to understand more clearly

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

- register file consists of 8 registers. 1 LR, 7 general purpose registers
- PC register is separated from the register file
- no wired connection between IDM & ALP
- temporary registers can be used
- The length for the data/instruction memory is 16 bits however registers are 8 bits. Also, operations are conducted with 8 bits
- R0, R1, R2, R3, R4, R5 are multi-purpose registers
- R7 is link register, it is in the register file.
- R6 is PC, PC is seperated from the register file



op

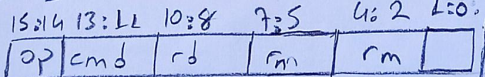
00 → DATA processing

01 → shift instructions

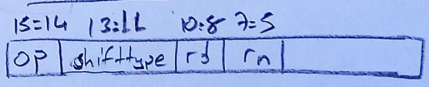
10 → memory instructions

11 → Branch instructions

DATA processing OP=00



shift inst. OP=01



cmd

000 → add rd, rn, rm

001 → addi rd, rn, MEM[inst 4:0]

010 → sub rd, rn, rm

011 → subi rd, rn, MEM[inst 4:0]

100 → and rd, rn, rm

101 → orr rd, rn, rm

110 → xor rd, rn, rm

111 → clr rd

shifttype "shift in & store at rd"

000 → rol rd, rn

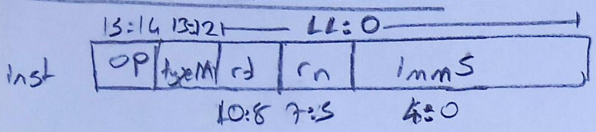
001 → ror rd, rn

010 → lsl rd, rn

011 → asr rd, rn

100 → lsr rd, rn

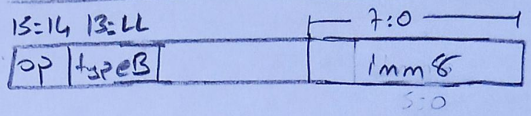
OP=10 memory instructions



typeM

- 00 → ldr rd, [rn, imm5]
- 01 → ldp rd, #inst7:0
- 10 → str rd, [rn, imm5]

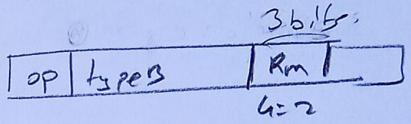
OP=11 Branch instructions



typeB

- 000 → B
- 001 → BL
- 010 → BI
- 011 → BEQ
- 100 → BNE
- 101 → BC
- 110 → BNC

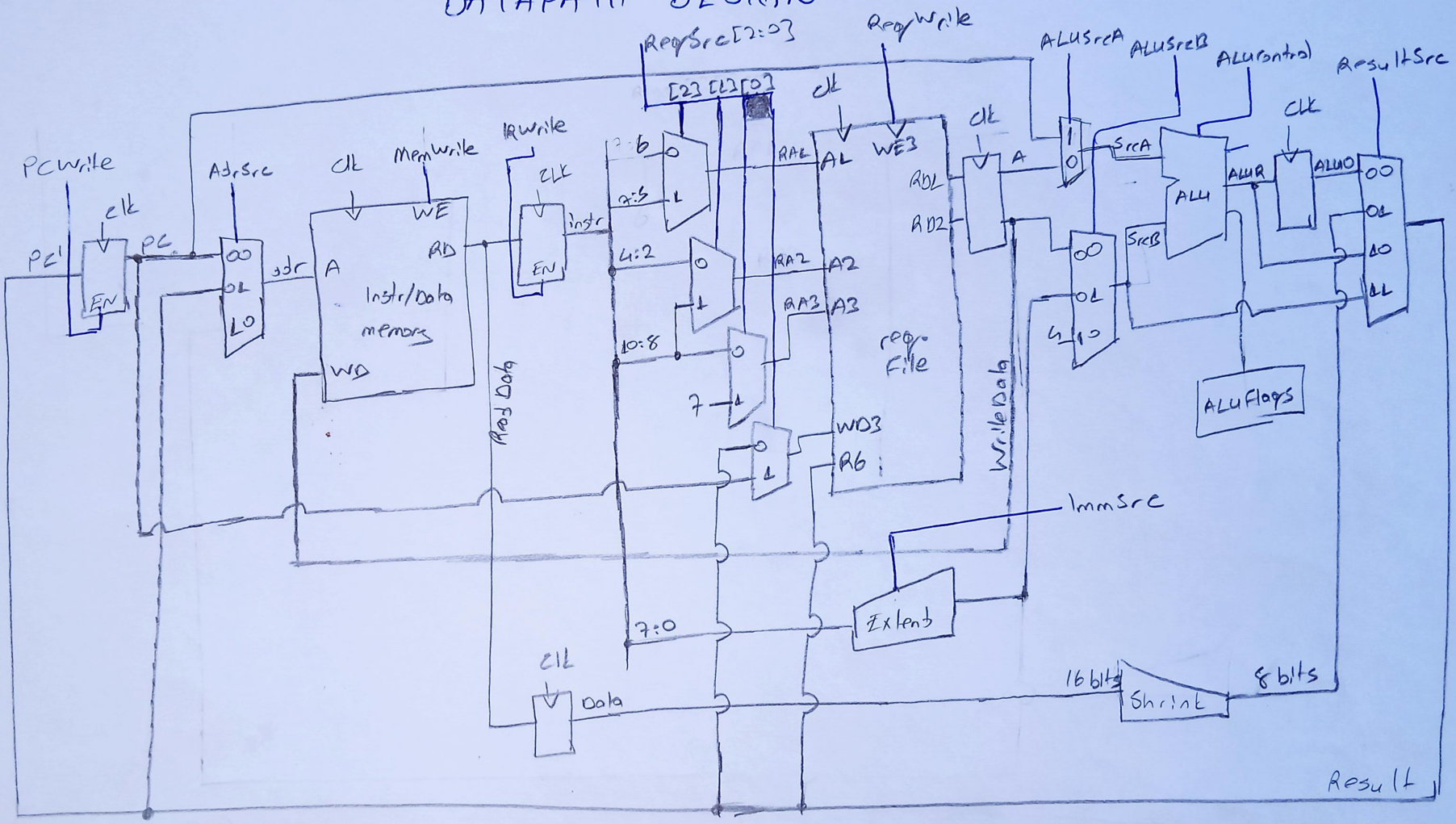
inst7:0 => imm8



- In indirect branch, the target address is specified indirectly either through memory or general purpose register
- Branch indirect takes "Rm" as its operand and causes a branch to address saved in Rm.
- In datapath ALU is updated, shift operations & data processing operations are conducted with the help of the ALU,

  ALU control control signal is provided from the control circuit.
- Just Data/Instruction memory word size is 16 bits. Other operation word sizes 8 bits.

DATAPATH DESIGN



1.2.7 Validation of Operation

(1) Explanation of each cycle & control signals

Instruction fetch (Cycle 1) (S0)

- Memory value pointed by the PC is loaded to Instruction register
- PC value is incremented by 4.

⇒ AdrSrc=00
⇒ PCWrite=1
⇒ ALUSrcA=1
⇒ ALUSrcB=10
⇒ ALUControl=4'b0000
⇒ ResultSrc=10
⇒ IRWrite=1

At the end of the clock cycle PC value incremented by 4.
M[PC] is in the instruction Regester end of the clock cycle

Decode (Cycle 2) (S1)

Rd, Rn, PC & are ready at Reg File. Inputs. Instruction in the instruction register is decoded.

- RegSrc 100
- PCWrite=0
- MemWrite=0
- IRWrite=0
- RegWrite=0
- ALUSrcA=1
- ALUSrcB=10
- ALUControl=00000(add)
- ResultSrc=10

- at the end of the clock cycle, values of the RD1, RD2 is loaded to the temporary registers.

- Data processing instructions => 4 cycles
  - Fetch, Decode, Execute, ALUWB
- Shift instructions => 4 cycles
  - Fetch, Decode, Execute, ALUWB
- Memory instructions: 3, 4 cycles or 5 cycles
  - LDI => 3 cycles: Fetch, Decode, ALUWBLDI
  - STR => 4 cycles: Fetch, Decode, MemAdr, MemWrite
  - LDR => 5 cycles: Fetch, Decode, MemAdr MemRead, MemWB
- Branch instructions
  - B, BL, BEQ, BNE, BC, BNC => 3 cycles
    Fetch, Decode, Branch
  - BLI => 3 cycles
    fetch, Decode, PCWrite
    - PCWrite: [Rm] value which is written in the register file written into PC.

Execute (Cycle 3 for Shift & Data) (S2)

- values of the temporary register is used for ALU operations

ALUSrcA=0
ALUSrcB=00
ALUop=1 (ALUControl from 0 to 11)
ResultSrc=00
RegWrite=0

the result is ready to register load at the end of the clock cycle.

ALUWB (cycle 4 for shifted Data) (S3)

- at the beginning of the clock cycle, in the entrance of the WD3, the result of the desired operation exists.

RegSrc[0]=0
RegWrite=1
ResultSrc=00

At the end of the clock cycle, data is written to specified register (Rd)

ALUWBLDI (cycle 3 for LDI instruction) (S4)

- ImmSrc=1, inst7:0 at the output of extend

The data value of the immediate bits directly loaded to Rd register.

ImmSrc=1
ALUSrcB=01
ResultSrc=11
RegSrc[0]=0
RegWrite=1

- load with immediate addressing is completed with that cycle. The data at the immediate part is loaded to the Rd register.

memAdr (cycle 3 for STR/LDR) (S5)

- the ImmS value in the instruction is added with Rn's value
- Then the added value is used to reach memory data

ImmSrc=0
ALUSrcB=01
ALUSrcA=0
ALUControl=0000

the memory addres calculated and loaded to temporary register
At the end of the clock cycle temporary register contains [Rn]+extend(imm)

MemWrite (Cycle 4 for STR) (56)

The stored value at the temporary register is selected with ResultSrc
Then value is stored to the memory.

ResultSrc=00
AdrSrc=01
MemWrite=1

(*) WD input comes from the temporary register

RegSrc in the decode stage must be 010 => For STR operation

At the end of the clock cycle Rd is loaded to MEM[Rn+imm5].

MemRead (Cycle 4 for LDR) (57)

with the calculated address in the (55) memory read occurs.

ResultSrc=00
AdrSrc=01
memWrite=0
the read data from the memory is loaded to DATA temporary register.

MemWB (Cycle 5 for LDR) (58)

the read value from the memory at the DATA temporary register beginning of the clock cycle
the data is loaded to Rd register at the end of the clock cycle with the help of the control signals.
Firstly 16 bits data is shrinked to the 8 bits data

ResultSrc=01
RegSrc[0]=0
RegWrite=1
} Rd is loaded with pointed value at the memory.

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

At the end of the clock cycle the calculated target address loaded to PC register

ALUSrcA=0

ALUSrcB=01

ALUcontrol=4'b0000 (add)

ResultSrc=10

Branch=1 then PCWrite=1

that means next cycle calculated branch target address is fetched

BL (cycle 3 for BL) (S10)

ALUSrcA=0

ALUSrcB=01

ALUcontrol=4'b0000 (add)

ResultSrc=10

RegWrite=1

Branch=1

Reg[0] = 1 while writing link register other procedures are the same as the "B".

BIPCwrite / BI (cycle 3 for BI) (SLL)

[Rm] value which is the value at the RD2's out temporary register is selected with ALUSrcB. Then, ResultSrc selects ALUSrcB content. Then, [Rm] value is written to PC with the help of the PCWrite.

ALUSrcB=00

ResultSrc=LL

PCWrite=1

(9) the end of the clock cycle [Rm] value is loaded to PC.

All in all;

(X) Branch instructions => 3 cycles

(X) memory instructions => 3, 4, 5 cycles (depends on the instruction)

(X) Shift & Data processing instructions => 4 cycles