1 Lab 4 - Pipelined LC-3b Microarchitecture

The following figures show a pipelined microarchitecture for the LC-3b ISA. You will use these diagrams to implement the pipelined version of LC-3b in Lab 4.

The pipeline for the LC-3b has five stages: Fetch (F), Decode/Register Read (DE), Address Generation/Execute (AGEX), Data Memory Access (MEM), and Store Result (SR). Between each stage, there are pipeline latches which are used to propagate data through the pipeline every cycle. Main logic components of each stage are shown on the figures. You will need to implement each stage as shown on these figures. Some logic blocks on the figures are labeled "LOGIC". The outputs of these blocks are shown in the figure, but the inputs to some of these blocks are omitted. It is your job to figure out the inputs to these blocks and figure out the implementation of the logic inside the blocks. All of these blocks, including the "Dependency Check Logic", generate outputs that control the stalling of the pipeline and insertion of pipeline "bubbles".

The remainder of this section describes the pipelined LC-3b microarchitecture.

1.1 An Overview of the Pipeline

LC-3b pipeline has five stages. F stage is used for fetching an instruction into the DE latches. In the DE stage, control store is accessed to generate some of the control signals required for the instruction. In parallel, with the control store access, register file is also accessed to retrieve the register operands of the instruction. The "dependency check logic" determines if the instruction in the DE stage is reading a value produced by an older instruction that is in the AGEX, MEM, or SR stage. If so, the instruction in the DE stage should not be propagated to the next stage. AGEX stage performs address computation for instructions that need to generate an address. Operate instructions also produce their result in this stage using the ALU and the shifter. Load instructions read the data memory in MEM stage. Store instructions write to the data memory in MEM stage. Control instructions update the PC when they reach MEM stage, because TRAP instruction needs to obtain the address from memory. The direction of a conditional branch instruction is also determined in MEM stage. Instructions that write into a destination register and set the condition codes perform these updates in the SR stage. ALL other instructions do nothing in the SR stage.

1.1.1 Handling of Dependencies

An instruction in the DE stage may require a value produced by an older instruction that is in the AGEX, MEM, or SR stage. If so, the instruction in the DE stage should not be propagated to the next stage. This dependency check is performed by the "Dependency Check Logic" in the DE stage. Inputs to this logic are shown on the figure for DE stage. The logic has one output signal, DEP.STALL. This signal is asserted if a dependency exists.

To find out whether a dependency exists, this logic block compares the destination register number of the instructions in AGEX, MEM, and SR stages to the source register numbers of the instruction in the DE stage. If a match is found and the instruction in DE actually needs the source register and the instruction in a later stage actually writes to the same register, DEP.STALL should be set to 1. The dependency check logic also checks the dependencies on condition codes. If the instruction in the DE stage is a conditional branch instruction (as indicated by the control signal BR.OP read from the control store), and if any of the instructions in AGEX, MEM, or SR stage is writing to condition codes, DEP.STALL should be set to 1.

1.1.2 Handling of Control Instructions

This basic pipelined microarchitecture of LC-3b stalls the pipeline when a control instruction is fetched. These instructions are resolved in MEM stage. Hence, a three-cycle bubble is inserted into the pipeline after each control instruction. A control instruction is identified using the BR.STALL signal stored in the microcontrol store. This signal is 1 for all control instructions, and 0 for other instructions. If any of the instructions in DE, AGEX, and MEM stages is a control inst. and it is valid, then the fetch stage should insert bubbles into the pipeline. This is accomplished by setting the valid bit of DE latches (DE.V) to 0.

1.1.3 Handling of Memory Operations

There are two caches in the pipelined LC-3b microarchitecture: Instruction Cache and Data Cache. Instruction Cache is accessed in the Fetch stage. If the data is found in the cache, the instruction cache asserts the ICACHE.R signal. Data Cache is accessed in the MEM stage by those instructions that need to read data from or write data into memory. These instructions have the DCACHE.EN control signal set in the control store. When the Data Cache access is complete, the data cache asserts the DCACHE.R signal.

1.2 Description of Pipeline Stages

1.2.1 Fetch Stage (F)

In this stage, instruction cache is accessed using the address in the PC. Instruction cache asserts the ICACHE.R signal when the access is complete. This signal is asserted some time in the middle of the clock cycle. If the I-cache is not ready, then logic in the fetch stage needs to stall the pipeline. To accomplish this, there are valid bits associated with pipeline latches. If the data fetched from the instruction cache is garbage (i.e. ICACHE.R is not asserted) and the pipeline is not stalling for some other reason, then the valid bit for the Decode stage latches (DE.V) should be set to 0. This valid bit indicates that the values in the DE latches are not meaningful and can be ignored.

In the simulator, you are provided an icache_access function. This function takes as input a 16-bit address, which should come from the PC. The outputs provided by this function are the 1-bit ICACHE.R signal and the 16-bit instruction. You need to use this interface to perform accesses to the I-Cache.

Fetch stage also includes the logic used to update the PC. If there are no stalls or control instructions in the pipeline, the PC should be incremented by 2. If a control instruction other than TRAP is supposed to write into PC, TARGET.PC value coming from MEM stage should be latched into the PC at the end of the current cycle. If a TRAP instruction is supposed to write into PC, TRAP.PC value coming from MEM stage should be latched into the PC at the end of the current cycle. The next value to be latched into the PC is controlled by the MEM.PCMUX signal which is generated in the MEM stage and the LD.PC signal which is generated by a logic block you should design.

At the end of the clock cycle, if LD.DE signal is asserted, DE latches (which contain DE.NPC, DE.IR, and DE.V) should latch their input values. DE.NPC contains the address of the next instruction. DE.IR contains the current instruction fetched from the instruction cache. It is your job to figure out the logic that generates the LD.DE signal. Think about when you don't want to load enable the DE latches. Hint: Do you want to load enable the DE latches if DEP.STALL is asserted? How about MEM.STALL?

1.2.2 Decode Stage (DE)

The instruction in DE stage accesses the control store. Control store has a 6-bit address. The address used to access the control store is obtained by concatenating IR[15:11] and IR[5]. Note that IR[11] and IR[5] are not actually part of the opcode, but they are used to access the control store. These bits are meaningful for some instructions (think about which instructions) and based on the value of these bits, different values are assigned to the control signals in different entries of the control store.

The control store contains 23 bits in each entry. These bits are shown in Table 2. These bits are used to control different structures in various stages of the pipeline. Table 2 shows the pipeline stages in which each bit in the control store is used.

At the end of the clock cycle, 20 bits from the control store are latched into the AGEX.CS latch.

The instruction in DE stage also reads the Register File and condition codes in DE stage. The register file has two read ports, one for SR1, one for SR2. DE.IR[8:6] is used to address the register file to read SR1. Either DE.IR[11:9] or DE.IR[2:0] are used to address the register file to read SR2. DE.IR[13] is used to select between DE.IR[11:9] or DE.IR[2:0]. Note that both register operands of an instruction are read in parallel in the same clock cycle unlike the LC-3b implementation you used for Lab 2.

At the end of the clock cycle, SR1 value from the register file is latched into the AGEX.SR1 latch, SR2 value is latched into the AGEX.SR2 latch. The value of the condition codes are latched into the 3-bit AGEX.CC latch. Condition code N is stored in AGEX.CC[2], Z is stored in AGEX.CC[1], and P is stored in AGEX.CC[0]. Note that condition codes and register file are read and the values obtained are latched regardless of whether or not an instruction needs these values.

The ID for the destination register is latched into the AGEX.DRID latch at the end of the clock cycle. DRMUX signal from the control store selects whether or not DE.IR[11:9] or 7 is latched into AGEX.DRID.

DE stage also contains the "Dependency Check Logic", whose operation is described above. This output of this logic (DEP.STALL) indicates whether or not the instruction in DE stage should be propagated forward. If DEP.STALL is asserted, the state of the DE latches should not be changed. A bubble needs to be inserted into the AGEX stage. This is accomplished by setting the valid bit for AGEX stage (AGEX.V) to 0. Other actions need to be taken to preserve the correct value of PC. Therefore DEP.STALL signal is also used by the structures physically located in the Fetch stage. It is your job to figure out how DEP.STALL signal affects the logic in Fetch stage.

One of the signals from the control store is the BR.STALL signal. This signal indicates that the instruction being processed is a control instruction and hence the frontend of the pipeline should be stalled until this instruction updates the PC in MEM stage. In DE stage, if DE.V is 1 and BR.STALL from control store is 1, DE.BR.STALL signal is asserted. This indicates that the instruction in the DE stage is a valid control instruction. DE.BR.STALL signal is used to insert bubbles into the pipeline in Fetch stage. It is again your job to figure out how to use the DE.BR.STALL signal. Think about all cases that might happen. For example, what happens if a branch instruction in DE stage is stalled due to a data dependency on an older instruction that sets the condition codes? Should the fetch stage still insert a bubble into the pipeline?

Note that you also need to design the logic required to generate LD.AGEX signal and input signal to the AGEX.V latch. Think about when you need to load disable the AGEX latches and when you need to insert a bubble into the AGEX stage.

1.2.3 Address Generation/Execute Stage (AGEX)

Operate instructions compute their results in this stage. Instructions that need to generate an address (to update the PC or to access data memory) also calculate their addresses in this stage. Muxes in this stage are controlled by signals read from the control store and then latched into the AGEX.CS latch.

Note that there are two logic structures you need to design in this stage. One logic structure determines the LD.MEM signal and the input signal to the MEM.V latch.

The other logic structure generates the signals to be sent to the previous stages of the pipeline. The outputs of this structure are V.AGEX.LD.CC, V.AGEX.LD.REG, and V.AGEX.BR.STALL. The first two signals are required by the dependency check logic. V.AGEX.BR.STALL indicated that the instruction being processed in the AGEX stage is a valid control instruction, and therefore the frontend of the pipeline needs to insert bubbles and stall. Note that this logic block simply gates the LD.CC, LD.REG, and BR.STALL control signals from the AGEX.CS latch with the AGEX.V bit.

1.2.4 Data Memory Access Stage (MEM)

In this stage, data cache is accessed by those instructions that need to read from or write into memory. Those instructions that need to read from or write into memory have their DCACHE.EN bit set in the control store. DCACHE.R/W bit in the control store indicates a read or write access. DATA.SIZE signal from the control store indicates whether this is a byte or word access.

The inputs to the Data Cache are:

1. 1-bit enable signal (DCACHE.EN signal from MEM.CS latch gated with MEM.V)

2. 2-bit WE signal. WE0 is the write-enable for the the low byte of a word. WE1 is the write-enable for the high byte of a word. You will need to generate these signals based on the values of DCACHE.R/W and MEM.ADDRESS[0].

3. 16-bit input address, MEM.ADDRESS[15:0], indicating which word is to be accessed.

4. 16-bit input data. This is the data that needs to be written into the cache if DCACHE.R/W is 1.

The outputs of the Data Cache are:

1. 1-bit DCACHE.R signal. If this signal is asserted it means that the access is complete.

2. 16-bit output data. This is the data read from the data cache. Note that you need design the logic to shift and sign extend the appropriate byte if the access is a byte access.

In the simulator, you are provided a dcache_access function which you need to call if the 1-bit enable signal to the Data Cache is set. This function takes as input the two WE signals, the 16-bit address, and 16-bit data. This function outputs the DCACHE.R bit and the 16-bit output data. You need to use this interface to perform accesses to the Data Cache, if it needs to be enabled. For further explanation, see the shell code provided.

One signal you need to generate in this stage is the MEM.STALL signal. This signal is used to stall the pipeline and insert a bubble into the SR latches if a valid memory access is not complete (i.e. DCACHE.R is 0). This signal is required by all previous stages in order to correctly stall the pipeline.

Control instructions update the PC when they reach the MEM stage. BR LOGIC shown on the figure for MEM stage generates the 2-bit PCMUX signal required by the Fetch stage of the pipeline. You should implement this logic. There are six inputs to this logic:

1. 1-bit valid bit from MEM.V latch. This bit is set if the instruction in MEM stage is valid.
2. 1-bit BR.OP signal from MEM.CS latch. This bit is set if the instruction is BR.
3. 1-bit UNCON.OP signal from MEM.CS latch. This bit is set if the instruction is JSR/JSRR or JMP.
4. 1-bit TRAP.OP signal from MEM.CS latch. This bit is set if the instruction is TRAP.
5. Values of the condition codes from the MEM.CC latch.
6. MEM.IR[11:9]: bits [11:9] of the instruction in MEM stage.

Another logic block you need to implement is the block that generates the input to the SR.V latch.

The last block you need to implement in this stage is the one whose outputs are V.MEM.LD.CC, V.MEM.LD.REG, and V.MEM.BR.STALL. V.MEM.LD.CC and V.MEM.LD.REG signals are inputs to the dependency check logic. V.MEM.BR.STALL signal is needed in the Fetch stage to insert a bubble into the pipeline. Note that, although control instructions are resolved in the MEM stage we still need to insert a bubble while a control instruction is being processed in the MEM stage. Think about why it should be this way. Also think about how LD.PC signal should be generated if a control instruction is being processed in MEM stage.

1.2.5 Store Result Stage (SR)

This stage is the stage where the instruction, if it is valid as indicated by SR.V, writes into the Register file and sets the condition codes. A 4-input mux, whose control signals come from the SR.CS latch, selects the 16-bit data to be written into the register file. This data can come from four places:

1. SR.ADDRESS latch, which contains the address generated in AGEX stage by the instruction and propagated through the pipeline.

2. SR.DATA latch, which contains the data read from memory (and shifted and sign-extended, if it was a byte access) in MEM stage and stored in SR.DATA at the end of the previous clock cycle.

3. SR.NPC latch, which contains the address of the next instruction. Think about why this would possibly be written into the destination register.

4. SR.ALU.RESULT latch, which contains the result generated by the shifter or the ALU in the AGEX stage and propagated through the pipeline.

You need to figure out which instructions need the data value from each of these locations to store it into the destination register. Accordingly, you should determine the value of the 2-bit DR.VALUEMUX control signal in the control store.

SR stage also contains the logic that determines the values of condition codes.

Last logic block contained in this stage outputs V.SR.LD.REG and V.SR.LD.CC signals indicating that a valid instruction is writing into the register file and condition codes respectively. These signals are used as write-enable signals in the register file and condition codes. They are also inputs to the dependency check logic.

Note that the latches for the SR stage do not have a load-enable bit (LD.SR) associated with them. Why is a load-enable signal not required for SR latches?

The structures in this stage are implemented in the simulator for you to get you started.

1.2.6 Control Signals and Their Propagation in the Pipeline

Control signals used in each stage are listed in Table 1. Note that some of these control signals come from the control store. The signals that come from the control store are shown in Table 2. There are 23 signals stored in each entry of the control store.

Three of these signals are only needed in DECODE stage. Therefore, only 20 of the control store signals need to be propagated to the next stage (AGEX). These signals are latched into the AGEX.CS latch shown on the datapath. Nine of the 20 signals are only needed in AGEX stage, so only 11 signals are propagated into the next stage. As shown in Table 2, 7 of the control signals fetched from the control store are not needed beyond MEM stage, so only 4 signals are latched into SR.CS latch. In the simulator, you are given the code that propagates control signals from one stage to the next.

1.2.7 Stall Signals

As mentioned in the previous sections, you will need to implement the logic to generate the following stall signals. These signals are also shown in Table 3.

1. ICACHE.R: Asserted if the I-Cache provides a useful instruction in this cycle. If 0, fetch stage should insert bubbles into the pipeline.

2. DEP.STALL: Asserted if the instruction in DE stage is valid and at least one of its input values has not been written into the register file yet.

3. MEM.STALL: Asserted if the instruction in MEM stage is valid and needs to access memory, and DCACHE.R signal is 0.

4. V.DE.BR.STALL: Asserted if the instruction in DE stage is a valid control instruction.

5. V.AGEX.BR.STALL: Asserted if the instruction in AGEX stage is a valid control instruction.

6. V.MEM.BR.STALL: Asserted if the instruction in MEM stage is a valid control instruction.

These signals are already declared and initialized in the simulator shell code for you. You will need to generate them and use them to implement some of the logic blocks in the pipeline that are left for you to implement.

1.3 What You Need to Do

In Lab 4, you need to complete the simulator to implement the pipeline described in this document. To be able to do this, you need to figure out what the unimplemented logic blocks are supposed to do.

You are not required to implement the RTI instruction or interrupt/exception handling.

Note that this pipeline does not execute JSRR R7 instruction correctly (In particular PC gets an incorrect value when this instruction is executed). Why is this? What small change would you make in the pipeline to fix this problem?

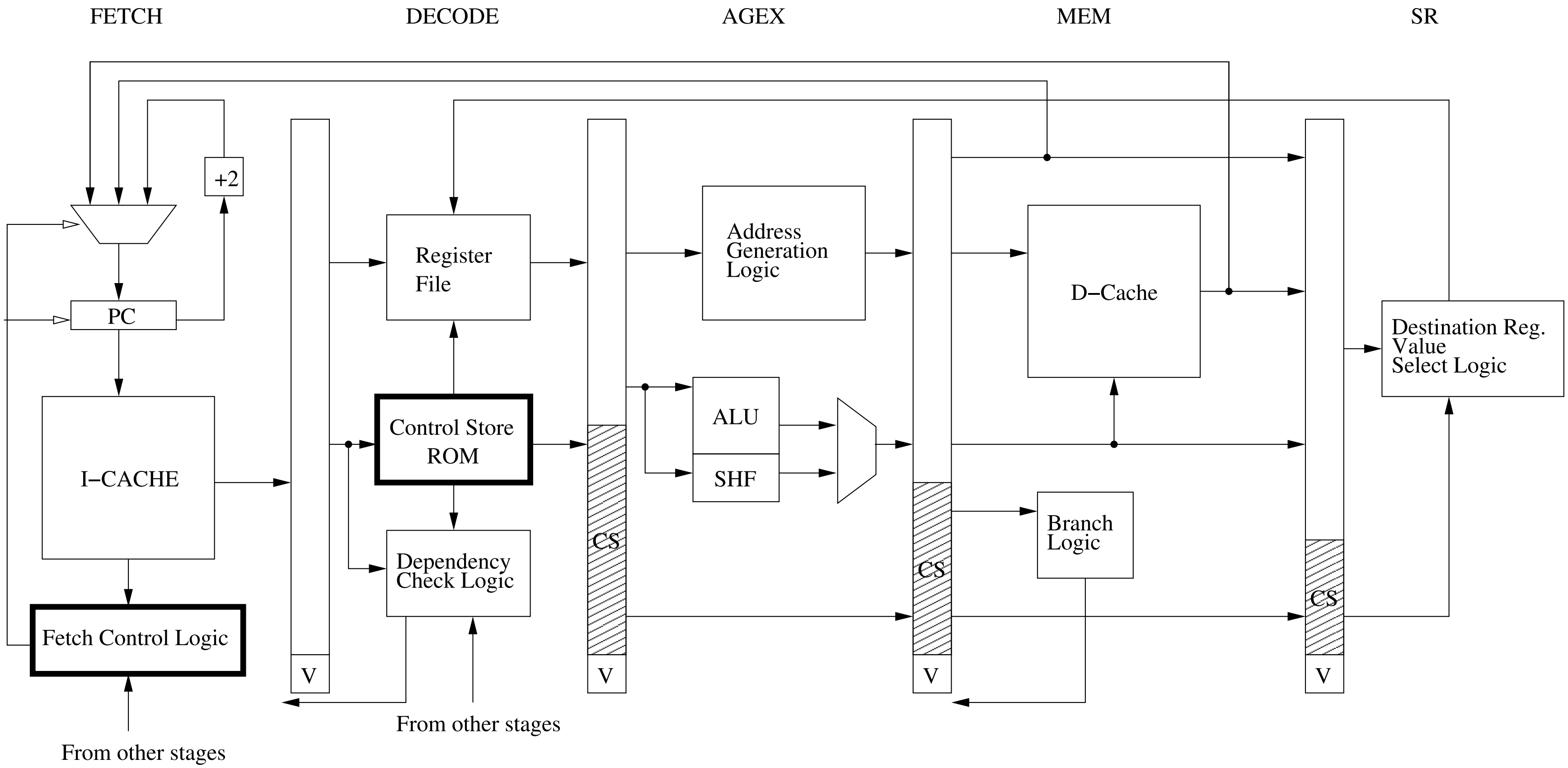

Fig.1 LC−3b pipeline diagram

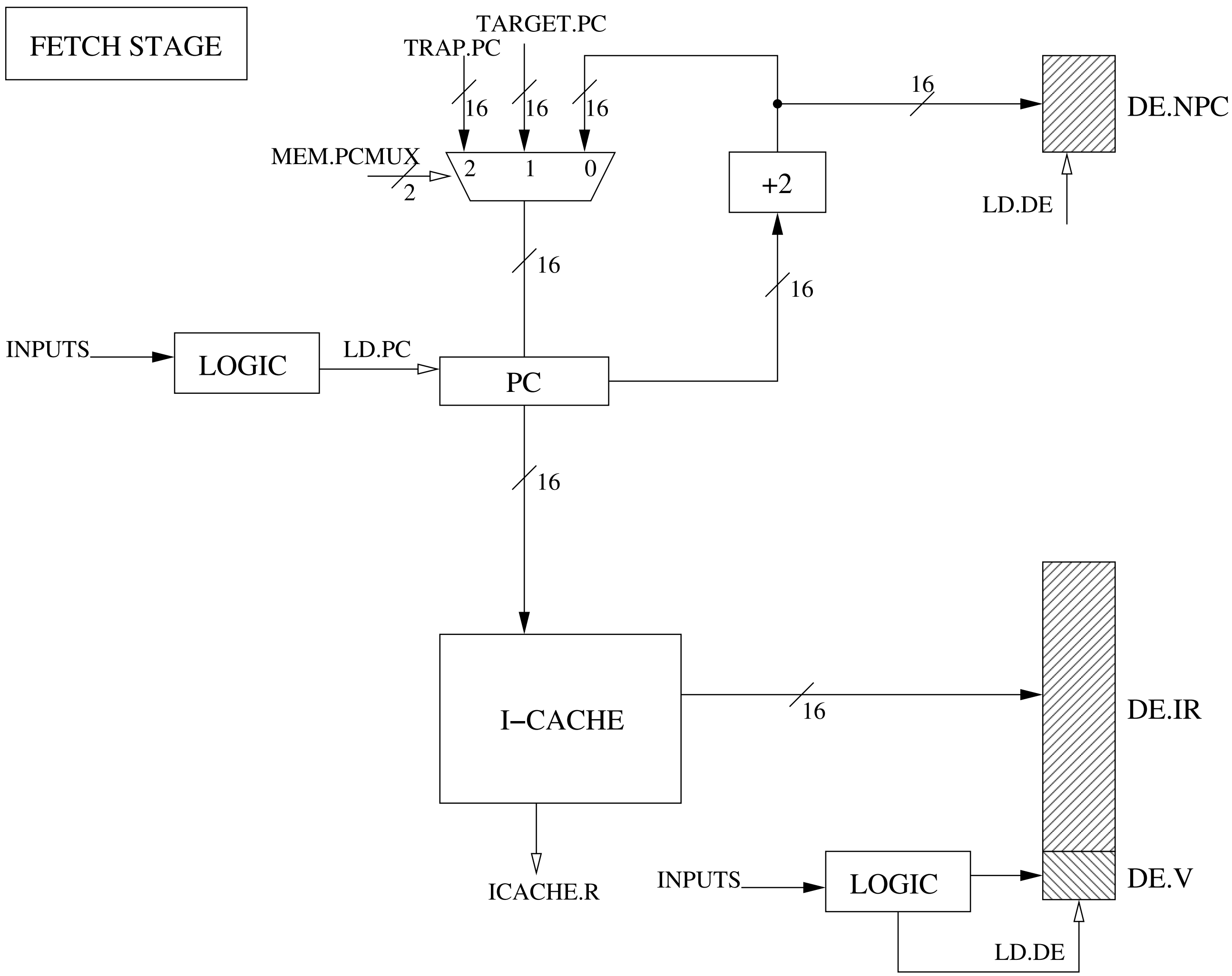

Fig.2 Fetch Stage (F−Stage)

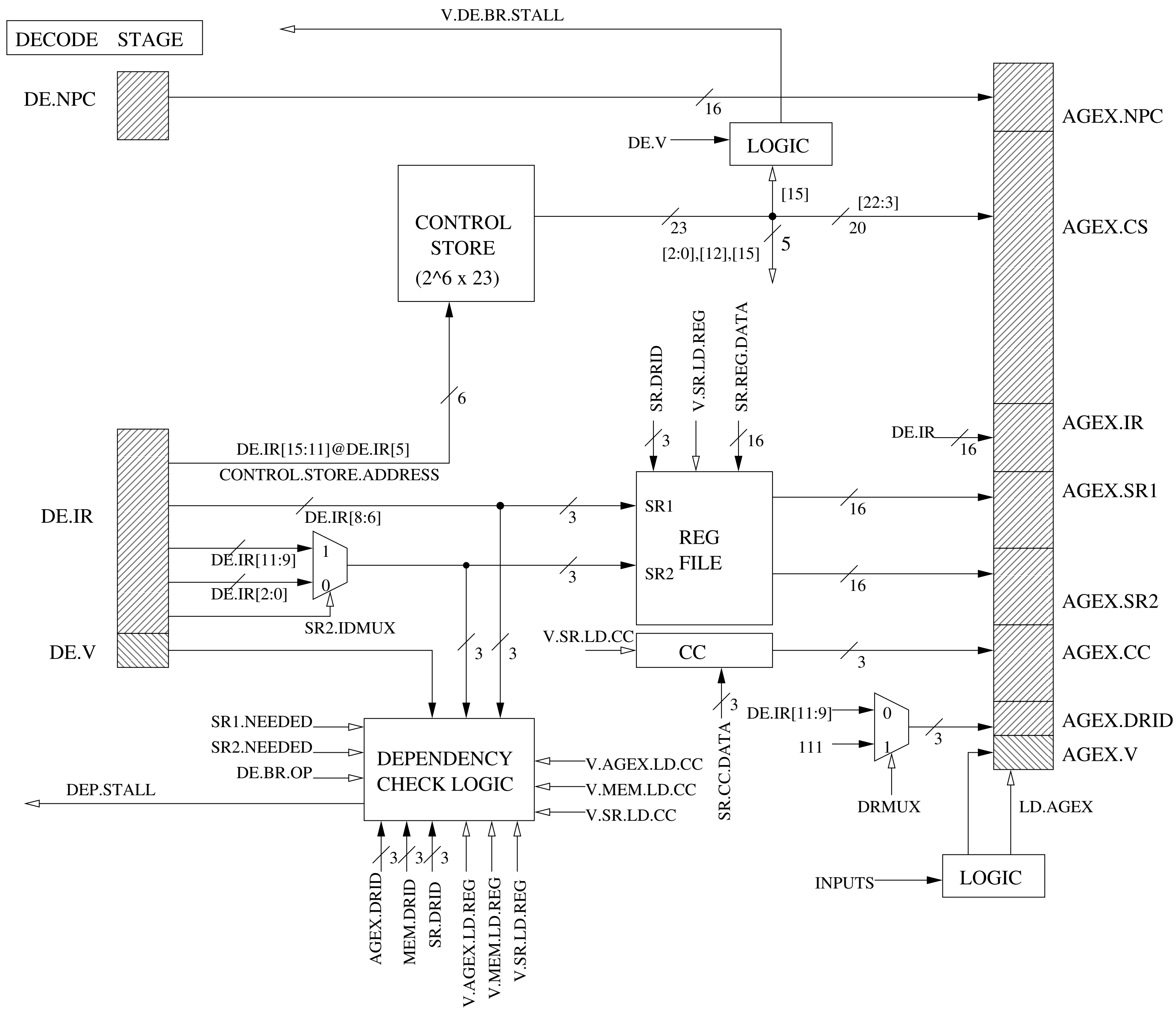

Fig.3 Decode Stage (DE−Stage)

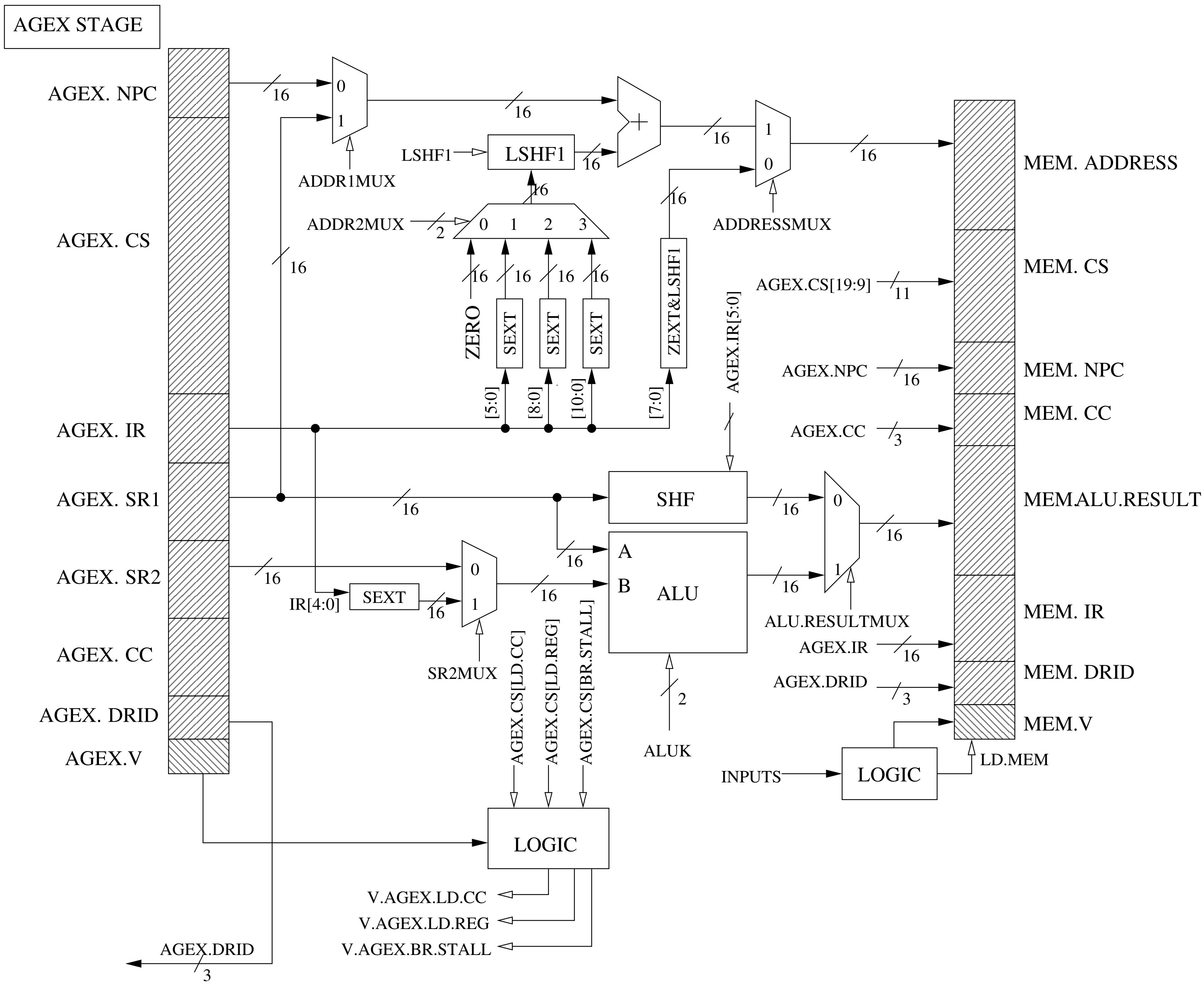

Fig.4 Address Generation Stage (AGEX–Stage)

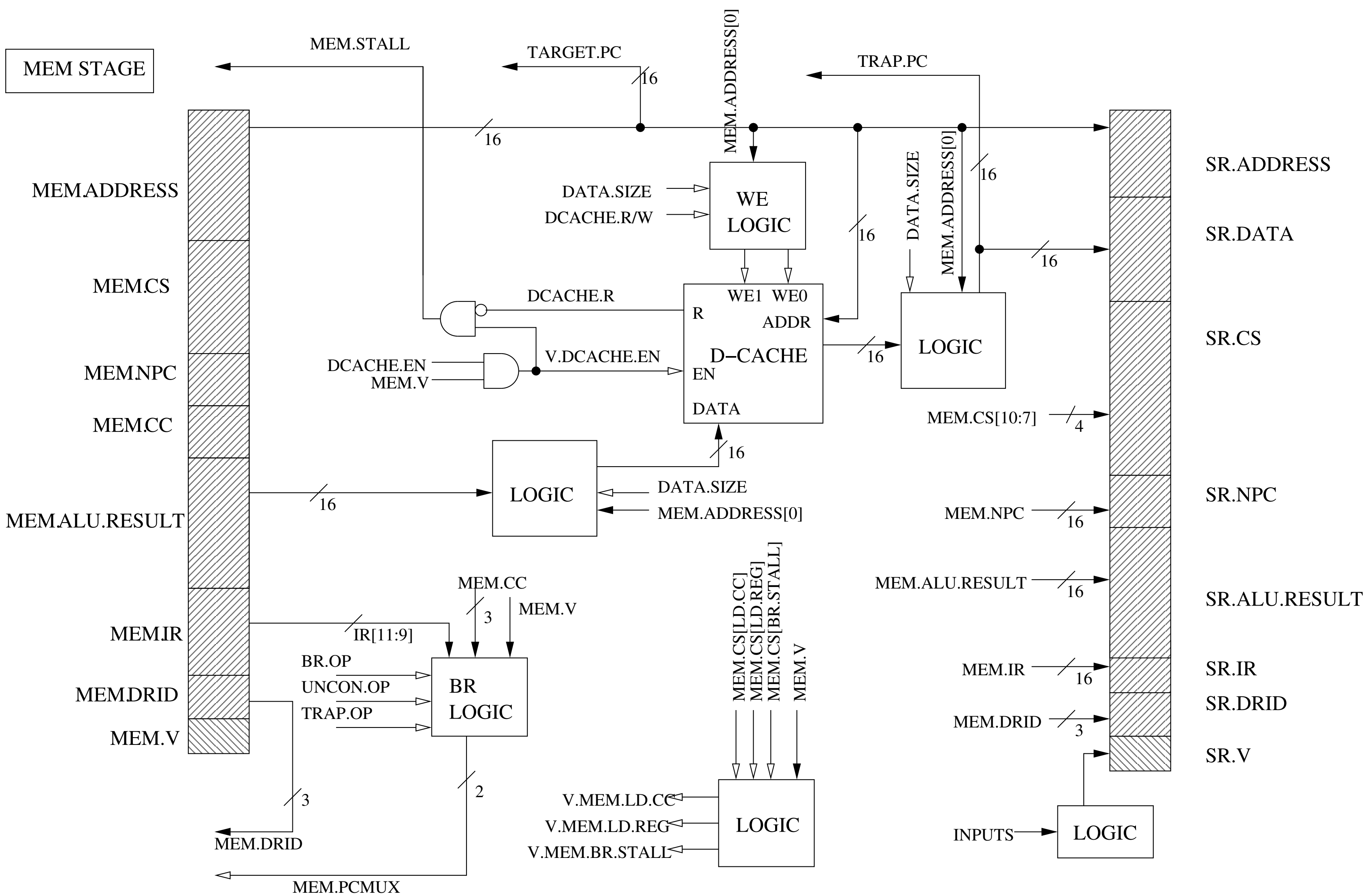

Fig. 5 Memory Stage (MEM−Stage)

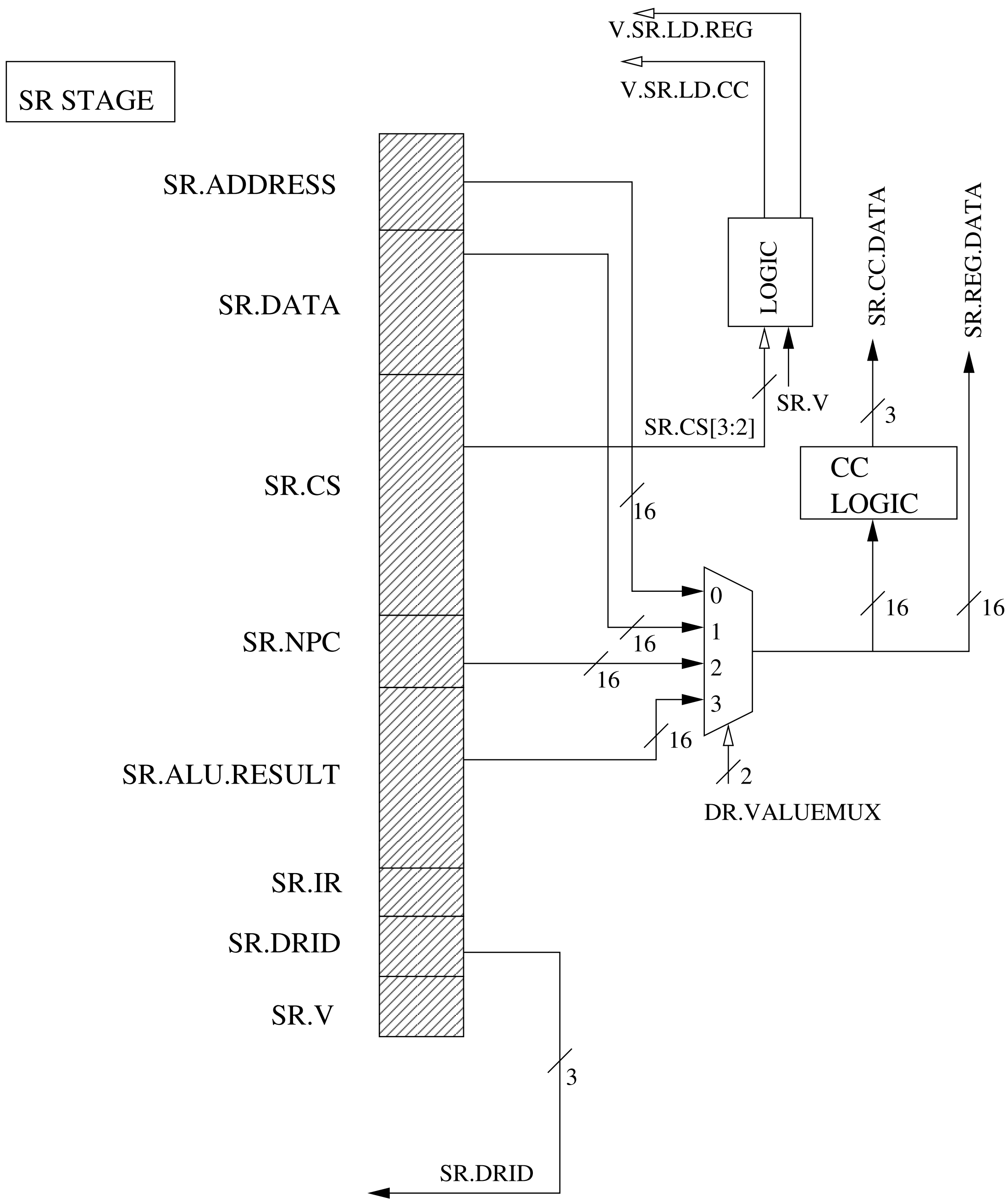

Fig.6 Store Result Stage (SR−Stage)

Stage	Signal Name	Signal Values	
FETCH	MEM.PCMUX/2:††	PC+2	;select pc+2
		TARGET.PC	;select MEM.TARGET.PC (branch target)
		TRAP.PC	;select MEM.TRAP.PC
	LD.PC/1:†	NO(0), LOAD(1)	
	LD.DE/1:†	NO(0), LOAD(1)	
DECODE	DRMUX/1:	11.9	;destination IR[11:9]
		R7	;destination R7
	SR1.NEEDED/1:	NO(0), YES(1)	;asserted if instruction needs SR1
	SR2.NEEDED/1:	NO(0), YES(1)	;asserted if instruction needs SR2
	DE.BR.OP/1:	NO(0), BR(1)	;BR Opcode
	SR2.IDMUX/1:†	2.0	;source IR[2:0]
		11.9	;source IR[11:9]
	LD.AGEX/1:†	NO(0), LOAD(1)	
	V.AGEX.LD.CC/1:††	NO(0), LOAD(1)	
	V.MEM.LD.CC/1:††	NO(0), LOAD(1)	
	V.SR.LD.CC/1:††	NO(0), LOAD(1)	
	V.AGEX.LD.REG/1:††	NO(0), LOAD(1)	
	V.MEM.LD.REG/1:††	NO(0), LOAD(1)	
	V.SR.LD.REG/1:††	NO(0), LOAD(1)	
AGEX	ADDR1MUX/1:	NPC	;select value from AGEX.NPC
		BaseR	;select value from AGEX.SR1(BaseR)
	ADDR2MUX/2:	ZERO	;select the value zero
		offset6	;select SEXT[IR[5:0]]
		PCoffset9	;select SEXT[IR[8:0]]
		PCoffset11	;select SEXT[IR[10:0]]
	LSHF1/1:	NO(0), 1bit Left shift(1)	
	ADDRESSMUX/1:	7.0	;select LSHF(ZEXT[IR[7:0]],1)
		ADDER	;select output of address adder
	SR2MUX/1:	SR2	;select from AGEX.SR2
		4.0	;IR[4:0]
	ALUK/2:	ADD(00), AND(01)	
		XOR(10), PASSB(11)	
	ALU.RESULTMUX/1:	SHIFTER	;select output of the shifter
		ALU	;select tput out the ALU
	LD.MEM/1:†	NO(0), LOAD(1)	
MEM	DCACHE.EN/1:	NO(0), YES(1)	;asserted if the instruction accesses memory
	DCACHE.RW/1:	RD(0), WR(1)	
	DATA.SIZE/1:	BYTE(0), WORD(1)	
	BR.OP/1:	NO(0), BR(1)	;BR
	UNCON.OP/1:	NO(0), Uncond.BR(1)	;JMP,RET, JSR, JSRR
	TRAP.OP/1:	NO(0), Trap(1)	;TRAP
SR	DR.VALUEMUX/2:	ADDRESS	;select value from SR.ADDRESS
		DATA	;select value from SR.DATA
		NPC	;select value from SR.NPC
		ALU	;select value from SR.ALU.RESULT
	LD.REG/1:	NO(0), LOAD(1)	
	LD.CC/1:	NO(0), LOAD(1)	

Table 1: Data Path Control Signals

†: The control signal is generated by logic in that stage

††: The control signal is generated by logic in another stage

Number	Signal Name	Stages
0	SR1.NEEDED	DECODE
1	SR2.NEEDED	DECODE
2	DRMUX	DECODE
3	ADDR1MUX	AGEX
4	ADDR2MUX1	AGEX
5	ADDR2MUX0	AGEX
6	LSHF1	AGEX
7	ADDRESSMUX	AGEX
8	SR2MUX	AGEX
9	ALUK1	AGEX
10	ALUK0	AGEX
11	ALU.RESULTMUX	AGEX
12	BR.OP	DECODE, MEM
13	UNCON.OP	MEM
14	TRAP.OP	MEM
15	BR.STALL	DECODE, AGEX, MEM
16	DCACHE.EN	MEM
17	DCACHE.RW	MEM
18	DATA.SIZE	MEM
19	DR.VALUEMUX1	SR
20	DR.VALUEMUX0	SR
21	LD.REG	AGEX, MEM, SR
22	LD.CC	AGEX, MEM, SR

Table 2: Control Store ROM Signals

Signal Name	Generated in	
ICACHE.R/1:	FETCH	NO, READY
DEP.STALL/1:	DEC	NO, STALL
V.DE.BR.STALL/1:	DEC	NO, STALL
V.AGEX.BR.STALL/1:	AGEX	NO, STALL
MEM.STALL/1:	MEM	NO, STALL
V.MEM.BR.STALL/1:	MEM	NO, STALL

Table 3: STALL Signals

SR1.NEEDED	SR2.NEEDED	DRMUX	ADDR1MUX	ADDR2MUX	LSHF1	ADDRESSMUX	SR2MUX	ALUK	ALU.RESULTMUX	BR.OP	UNCON.OP	TRAP.OP	BR.STALL	DCACHE.EN	DCACHE.RW	DATA.SIZE	DR.VALUEMUX	LD.REG	LD.CC	
																				000000 (0)
																				000001 (1)
																				000010 (2)
																				000011 (3)
																				000100 (4)
																				000101 (5)
																				000110 (6)
																				000111 (7)
																				001000 (8)
																				001001 (9)
																				001010 (10)
																				001011 (11)
																				001100 (12)
																				001101 (13)
																				001110 (14)
																				001111 (15)
																				010000 (16)
																				010001 (17)
																				010010 (18)
																				010011 (19)
																				010100 (20)
																				010101 (21)
																				010110 (22)
																				010111 (23)
																				011000 (24)
																				011001 (25)
																				011010 (26)
																				011011 (27)
																				011100 (28)
																				011101 (29)
																				011110 (30)
																				011111 (31)
																				100000 (32)
																				100001 (33)
																				100010 (34)
																				100011 (35)
																				100100 (36)
																				100101 (37)
																				100110 (38)
																				100111 (39)
																				101000 (40)
																				101001 (41)
																				101010 (42)
																				101011 (43)
																				101100 (44)
																				101101 (45)
																				101110 (46)
																				101111 (47)
																				110000 (48)
																				110001 (49)
																				110010 (50)
																				110011 (51)
																				110100 (52)
																				110101 (53)
																				110110 (54)
																				110111 (55)
																				111000 (56)
																				111001 (57)
																				111010 (58)
																				111011 (59)
																				111100 (60)
																				111101 (61)
																				111110 (62)
																				111111 (63)