

به منظور ایجاد کمترین تغییرات در مسیر داده، قالب دستورات مورد نظر به شکل زیر انتخاب شده است:

opcode[6]						11101 (5'd29)					i [5]		Don't Care [16]														
31						26		25		21		20		16		15		0									

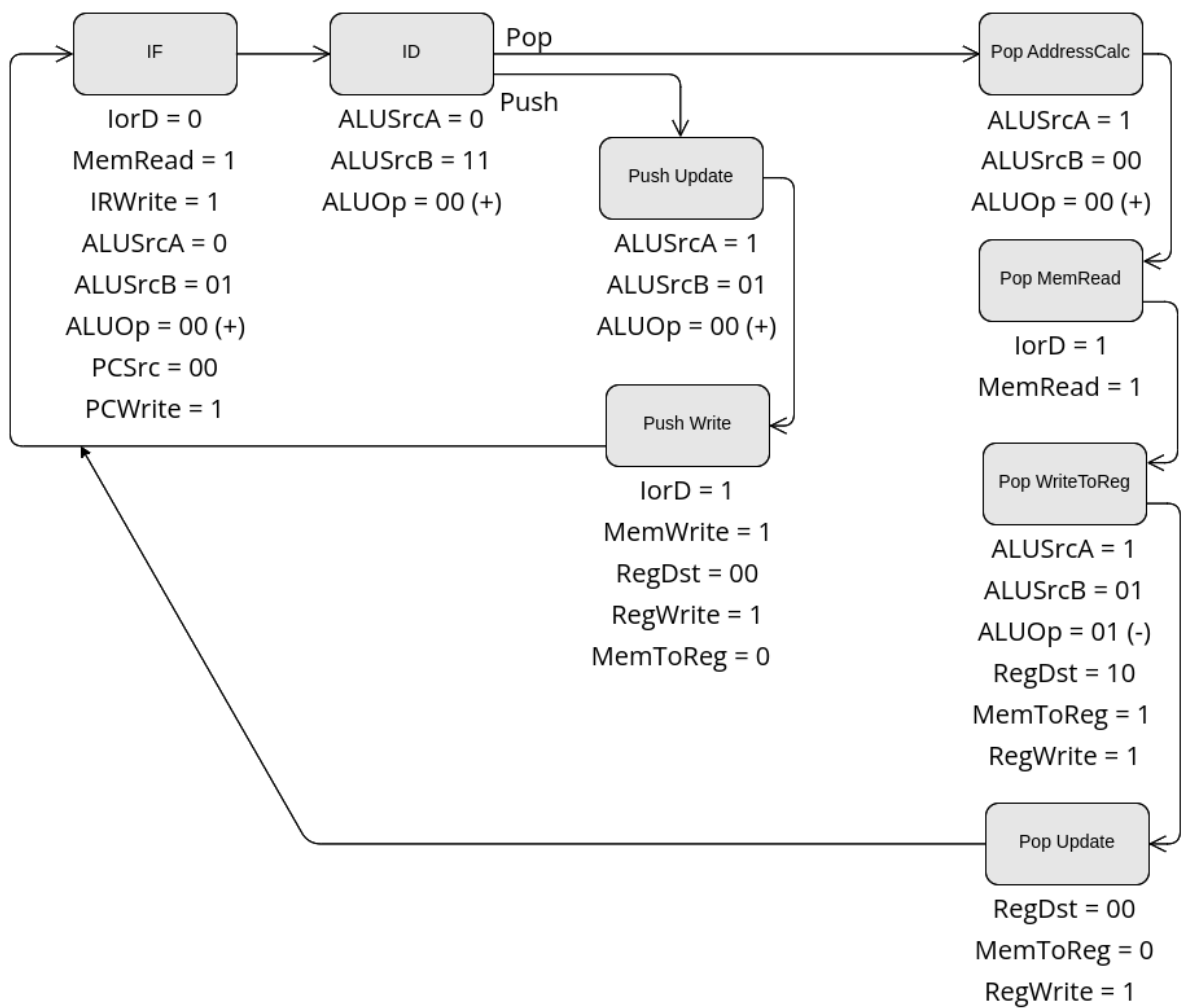
opcode[6]		11101 (5'd29)				00000 (5'd0)				i [5]		Don't Care [11]					
31	26	25	21	20	16	15	11	10							0		

The diagram illustrates the internal components and data flow of a computer system, likely a MIPS-style processor. Key components and their interactions are as follows:

- PC (Program Counter):** Receives **PCWrite** and **PCWrite Cond** signals. Its output is connected to a 32-bit multiplexer (MUX) and the **Memory** block.
- Memory:** Receives **Address** and **Write Data** from the **IR** block. It outputs **Read Data** to the **MDR** (Memory Data Register).
- IR (Instruction Register):** Receives **IRWrite** and **MemToReg** signals. It outputs **RegDst** (25-21), **RegWrite** (20-16), and **MemToReg** (15-11) signals to the **Register File**.
- MDR (Memory Data Register):** Receives **Read Data** from **Memory** and outputs **MemToReg** (15-11) to the **Register File**.
- Register File:** Contains multiple registers (#1, #2, ..., #w). It receives **RegDst** and **RegWrite** signals. It outputs **Read Data1** and **Read Data2** to the **ALU** and **Sign Extend** block.
- ALU (Arithmetic Logic Unit):** Receives **ALUSrcA** (31-28) and **ALUSrcB** (31-28) signals. It outputs **ALU Result** and **ALU control** signals.
- Sign Extend:** Receives **Read Data1** and **Read Data2** from the **Register File** and outputs **Sign Extend** (32-bit) to the **ALU**.
- Shift Registers (Shl 2):** Receives **Shl 2** (28-bit) and **Shl 2** (32-bit) signals. It outputs **Shl 2** (28-bit) and **Shl 2** (32-bit) signals to the **ALU**.
- ALU Out:** Receives **ALU Result** and **ALU control** signals. It outputs **ALU Out** (32-bit) to the **PC** and **Shl 2** block.
- PCSrc:** Receives **PCSrc** (32-bit) and **PCSrc** (28-bit) signals. It outputs **PCSrc** (32-bit) to the **PC**.

	memWrite	memRead	lorD	IRWrite	regDst	memToReg	regWrite	ALUSrcA	ALUSrcB	ALUOp	PcSrc	PcWrite	PcWriteCond
Instruction Fetch (IF)	0	1	0	1	—	—	0	0	01 (4)	00 (+)	00 (ALU)	1	0
Instruction Decode (ID)	0	0	—	0	—	—	0	0	11 (Shl2)	00 (+)	—	0	0
Push Update	0	0	—	0	—	—	0	1	01 (4)	00 (+)	—	0	0
Push Write	1	0	1	0	00	0	1	—	—	—	—	0	0
Pop AddressCalc	0	0	—	0	—	—	0	1	00 (B)	00 (+)	—	0	0
Pop MemRead	0	1	1	0	—	—	0	—	—	—	—	0	0
Pop WriteToReg	0	0	—	0	10	1	1	1	01 (4)	01 (-)	—	0	0
Pop Update	0	0	—	0	00	0	1	—	—	—	—	0	0

نمودار حالت واحد کنترل:



پرسش 2:

پیاده‌سازی به روش Micro-Memory:

در Micro-Memory، هر ریز دستور یک سطر در حافظه می‌شود که هر کدام 40 بیت برای تعیین سیگنال‌های کنترلی نیاز دارد.

پس در این نوع پیاده‌سازی، $20000 = 40 \times 500$ بیت فضا اشغال می‌شود. اما از آنجایی که برای پیاده‌سازی از ROM استفاده کردیم، تعداد سطرها باید توانی از 2 باشند و در نتیجه به 512 سطر نیاز داریم که 12 تا از آن خالی می‌ماند.

کل فضای مورد نیاز برابر با $20480 = 40 \times 512$ بیت خواهد بود.

پیاده‌سازی به روش Nano-Memory:

در Nano-Memory، هر ریز دستور یکتا، یک سطر در حافظه نانو می‌شود که هر کدام 40 بیت برای تعیین سیگنال‌های کنترلی نیاز دارد.

پس در این نوع پیاده‌سازی، حافظه نانو $8000 = 40 \times 200$ بیت فضا اشغال می‌کند. اما از آنجایی که برای پیاده‌سازی از ROM استفاده کردیم، تعداد سطرها باید توانی از 2 باشند و در نتیجه به 256 سطر نیاز داریم که 56 تا از آن خالی می‌ماند.

کل فضای مورد نیاز حافظه نانو برابر با $10240 = 40 \times 256$ بیت خواهد بود.

از طرفی در حافظه میکرو مانند حالت قبل، 512 سطر خواهیم داشت که هر کدام 8 بیت برای آدرس‌دهی حافظه نانو نیاز دارند. پس حافظه میکرو به $4096 = 8 \times 512$ بیت نیاز دارد.

در نتیجه کل حافظه مصرفی در این نوع پیاده‌سازی برابر با $14336 = 4096 + 10240$ بیت خواهد بود.

میزان صرفه‌جویی در حافظه در این نوع پیاده‌سازی، برابر با $6144 = 20480 - 14336$ بیت خواهد بود.