Lecture 31 – Processor design 6

Dr. Aftab M. Hussain,

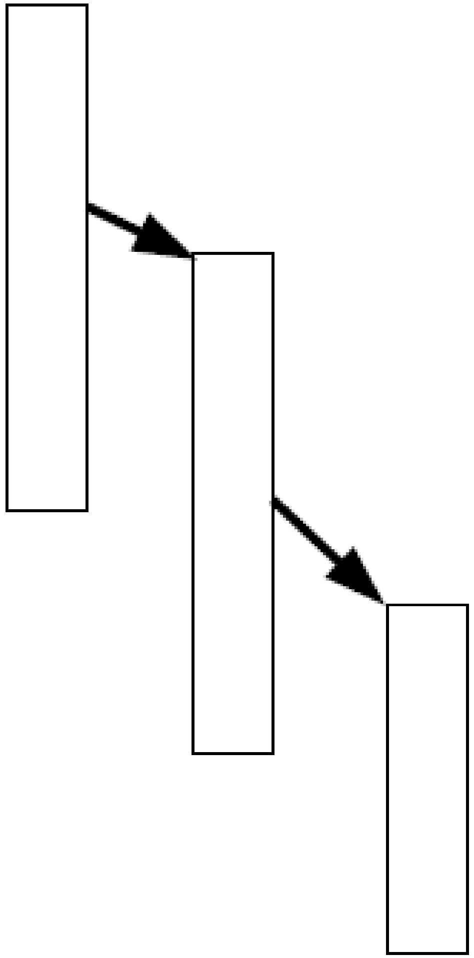
Assistant Professor, PATRIoT Lab, CVEST

Jump instructions

- We will now look at the remaining few instructions to make our processor more complete
- We had no branching instructions so far; all programs have to be a strictly linear sequence of instructions
- That is obviously not a very desirable situation
- We need the capability to branch or to break the sequential flow of instructions to implement any sort of loops
- Additionally, we need the capability to branch based on some condition based on the values of registers
- We use two forms of branching: one based on an immediate address and the other based on a register value (flag register)

Jump instructions

- Branching involves the shifting of the program execution from one point in the program to another
- This is a change in the control flow of the program and may be used to perform different actions on the basis of the results achieved so far
- Jump is a type of branching where the control is transferred absolutely, without any memory of the branching point
- Branching is natural in our every day activities



The flag register

- The branching may be conditional, based on a current state of the processor
- We include a flag register in the processor to indicate particular conditions
- The condition of one of the flag register bits can be used for branching
- If the jump is conditioned on a flag bit being set, the branching happens only if that flag has a value of 1 and the program proceeds with the instruction at the branch address
- If the flag is at state 0, execution proceeds normally with the next instruction as if the conditional branch instruction is a *nop* instruction
- The flag register stores limited history of the results of the computations performed by the processor
- This may include aspects like: Did the last arithmetic operation result in an overflow? Did it result in a carry from the most significant bit? Was the result of the previous operation a zero?
- These, in conjunction with branching, are essential to control the program based on the results of operations

The flag register

- For example, if we want to run a loop ten times, we can repeat the code 10 times, which makes the code long
- It also allows no flexibility to run the code 12 times, if we desire it
- An alternative is to use a count (typically stored in a register) that is initialized to 10
- After one set of computations is over, the count can be reduced by 1
- The program can branch to the start of the computations if the count is still not zero
- It is clear that the second option results in shorter code
- Even better, if the count is initialized to 12 or 25, the code remains exactly the same

The flag register

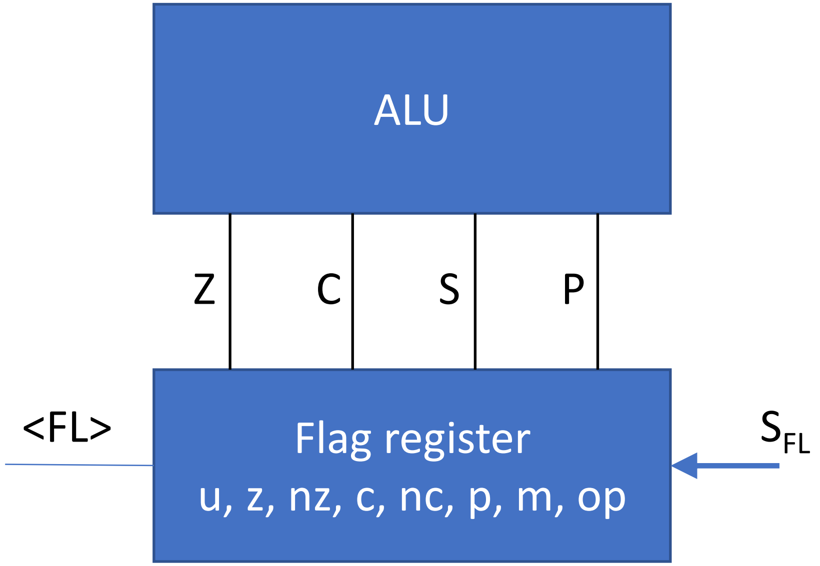
- Our simple processor has the following 4 flag bits: zero, carry, sign, and parity, with respective flags Z, C, S, and P
- The zero flag is set if the previous ALU operation produced an exact 0 as the result (i.e., if AR is zero)
- Similarly, the carry flag is set if the previous operation resulted in a carry-out or borrow-in from the most significant bit
- The S bit copies the sign bit of the last arithmetic operation and becomes 1 if the result was negative
- The parity bit counts the number of 1 bits in the result of the last operation
- If that number is odd, the parity bit is 1

The flag register

- Instructions that do not use the ALU – such as the data movement instructions, branching instructions, and the like – do not change the flag values
- Some processors group all flags into a special register word known as the Program Status Word (PSW)
- Special instructions may move the PSW to or from internal registers or memory
- This allows their manipulation as data

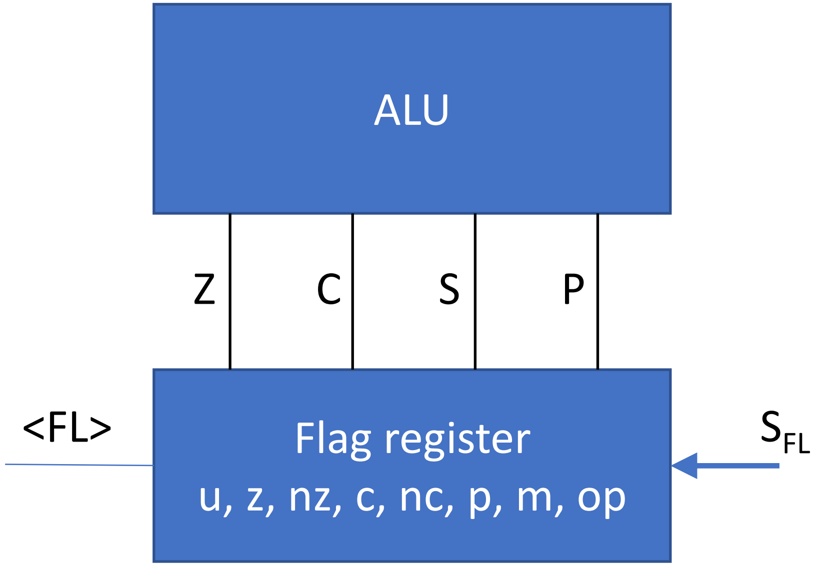
Jump instruction

- For the simple flag register defined, the <FL> flag for conditional instructions can take one of the following values: u, z, nz, c, nc, p, m, op
- These respectively stand for unconditional, zero, non-zero, carry, no-carry, positive, minus, and odd-parity
- Unconditional case is always true, irrespective of the state of the flag bits
- Zero condition is true when the Z bit is set and the non-zero condition is true otherwise
- Similarly, the carry and no-carry conditions are true when the C bit of the flags is 1 and 0 respectively
- The plus condition is true when the sign bit S is 0 and the minus condition is true otherwise
- The odd-parity condition is true if the flag bit P is 1



Implementing the jump instruction

- To implement the jump function, we need to change the PC to the contents of the AR if the flag condition is satisfied
- If not, it should have no effect
- The conditional jump instructions use a modified control signal, labelled "*End if <FL>'*"
- This means that the End control signal is activated only if the selected flag is at a 0 level
- This is the case where the condition is not satisfied and hence the instruction has no impact

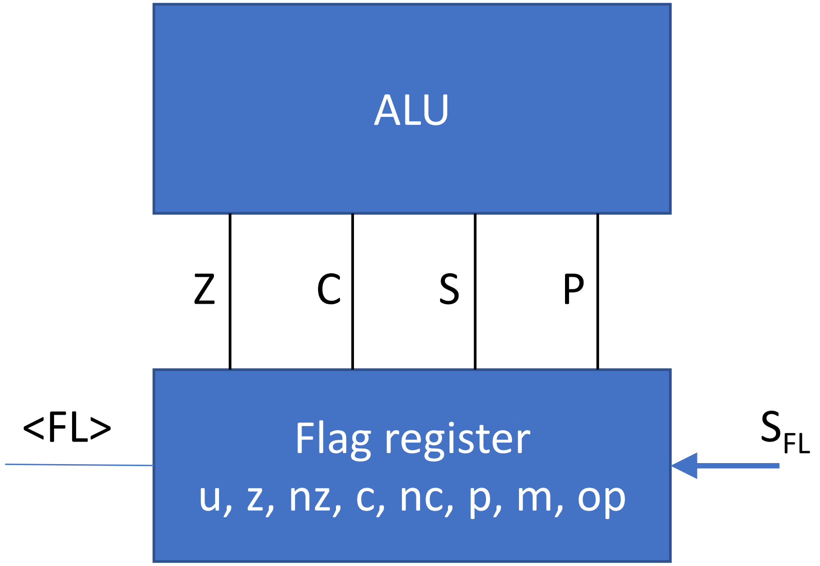


Assembly Instruction	Machine Code	Action
jmpd<FL> xx	E0-E7	[PC] ← xx if <FL> = 1
jmpr<FL>	E8-EF	[PC] ← [AR] if <FL> = 1

Instruction	Control Signals	Select Signals
jumpd<FL> xx	Ck 3: E_{PC}, L_{MR}, I_{PC}, E_{FL}, End if <FL>' Ck 4: RD, L_{PC}, End	S_{FL} ← <FL> -
jmpr<FL>	Ck 3: E_{FL}, End if <FL>' Ck 4: E_{AR}, L_{PC}, End	S_{FL} ← <FL> -

Implementing the jump instruction

- However, for instructions with immediate operands (such as jumpd), the next word has to be jumped over so that the PC points to the next real instruction
- At the same time, the value of PC should be incremented whether the value of flag is true or not
- Hence, the PC value is saved in MR and PC is incremented
- Then if the flag condition is satisfied, the value at the address is loaded into PC



Assembly Instruction	Machine Code	Action
jmpd<FL> xx	E0-E7	[PC] $\leftarrow$ xx if <FL> = 1
jmpr<FL>	E8-EF	[PC] $\leftarrow$ [AR] if <FL> = 1

Instruction	Control Signals	Select Signals
jumpd<FL> xx	Ck 3: E_{PC}, L_{MR}, I_{PC}, E_{FL}, End if <FL>' Ck 4: RD, L_{PC}, End	$S_{FL} \leftarrow$ <FL> -
jmpr<FL>	Ck 3: E_{FL}, End if <FL>' Ck 4: E_{AR}, L_{PC}, End	$S_{FL} \leftarrow$ <FL> -