LECTURE 4

Dynamic Scheduling (continued):

Each reservation station holds an instruction that has been issued and is awaiting execution at a function unit. It also contains the operand values for that instruction (if they have already been completed) OR the names of the reservation stations that will provide the operand values.

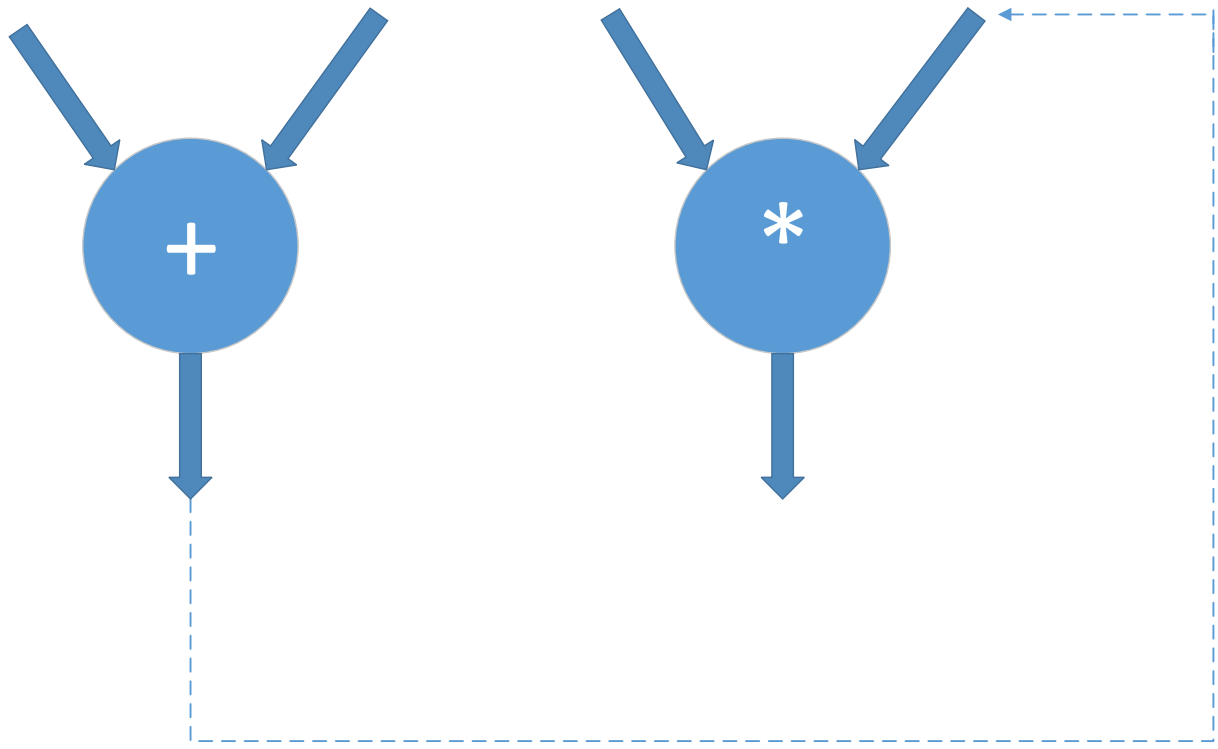
Load Buffers:

i) Hold the components of the effective address until it is computed
ii) Track outstanding loads that are waiting in the memory
iii) Hold the results of completed loads that are waiting for the CDB

Store Buffers:

i) Hold components of the effective address
ii) Hold addresses of outstanding stages
iii) Hold addresses and values until memory is available

All results from the functional units and from memory are sent on the **common data bus (CDB)**; which goes everywhere except to the load buffer.



STEPS IN EXECUTION OF AN INSTRUCTION:

1. ISSUE:

   Get the next instruction from the head of the instruction queue, which is maintained in FIFO order.

   If there is an empty reservation station:
   - Issue the instruction to the station
   - Operands: If in registers, issue to reservation station,
     Else keep track of all functional units that will produce the operands

   Else there is a structural hazard and the instruction stalls until a station or buffer is freed.

2. EXECUTE:

   If one or more of the operands is not yet available,
   Monitor the common data bus while waiting for it to be computed. When an operand becomes available, it is placed into corresponding reservation station.
   Else (*all operands are available*)
   Execute operation at corresponding functional unit.

LOADs and STOREs require a two-step execution process. The first step computes the effective address when the base register is available, and the effective address is then placed in the load or store buffers.

LOADs in the load buffer execute as soon as the memory unit is available.

STOREs in the store buffer wait for the value to be stored before being sent to the memory unit.

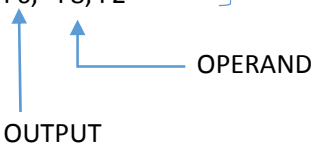
3. WRITE

   When the result is available, write it in the CDB and from there into the registers and into any reservation stations (including store buffers) waiting for this result.

   STOREs also write data to memory during this step: When both the address and data are available, they are sent to the memory unit and the store completes.

Dynamic Scheduling: *(example)*

1. L.D D6, 34 (R2) completed ; result on CDB
2. L.D F2, 45 (R3)
3. MUL.D F0, F2, F4
4. SUB.D F8, F2, F6 issued
5. DIV.C F10, F0, F6
6. ADD.D F6, F8, F2



NT:
SUB TAKEN AS ADD
DIV TAKEN AS MULT

RESERVATION STATIONS

NAME	BUSY	OP	V_j	V_k	Q_j	Q_k	A
Load 1	No						
Load 2	Yes	Load					45 + Regs(R3)
Add 1	Yes	SUB		Mem [34 + Regs [R2]]	Load 2		
Add 2	Yes	ADD			Add 1	Load 2	
Add 3	No						
Mult 1	Yes	MUL		Regs [F4]	Load 2		
Mult 2	Yes	DIV		Mem [34 + Reg [R2]]	Mult 1		

Notation:

Each reservation station has seven fields:

V_j, V_k : The value of the source operands
Q_j, Q_k: The reservation stations that will produce the source operand
A: Used to hold information for the memory address calculation for a load or store. Initially, the immediate field of the instruction is stored here; after the address calculation, the effective address is stored here.
Busy: Indicates that this reservation station and its accompanying functional unit are occupied
Op: The operation to perform on source operands

The register file has a field Q:

Q_i : The number of the reservation station that contains the operation whose result should be stored into this register.

Register Status

REGISTER	Q_i
F0	Mult 1
F2	Load 2
F6	Add 2
F8	Add 1
F10	Mult 2

TOMASULO'S ALGORITHM:

A key approach to allow execution to proceed in the presence of dependencies was used by IBM360/91 floating point unit.

Invented by Robert Tomasulo, this scheme tracks when operands for instructions are available, to minimize RAW hazards, and introduces register renaming, to minimize WAW and WAR hazards

Register renaming is provided by the reservation stations, which buffer the operands of the instructions waiting to issue, and by the issue logic. The basic idea is that a reservation stations fetches and buffers an operand as soon as it is available, eliminating the need to get the operand from a register. In addition, pending instructions designate the reservation station that will provide their input. Finally, when successive writes to a register overlap in execution, only the last one is actually used to update the register. As instructions are issued, the register specifies for pending operands are renamed to the names of the reservation station, which provides register renaming.

Instruction state	Wait Until	Action or book-keeping

Issue	Station 'r' empty	If(registerStat[rs].$Q_i \neq 0$){
		RS[r].$Q_j \leftarrow$ RegisterStat[rs].Q_i
		}
		Else{
		RS[r].$V_j \leftarrow$ Regs[rs];
		RS[r].$Q_j \leftarrow 0$
		}
		If(RegisterStat[rt].$Q_i \neq 0$){
		RS[r].$Q_k \leftarrow$ RegisterStat[rt].Q_i
		}
		Else{
		RS[r].$V_k \leftarrow$ Regs[rt];
		RS[r].$Q_k \leftarrow 0$
		}
		RS[r].Busy $\leftarrow$ Yes;
		RegisterStat[rd].Q_i = r;

r : Reservation Station Number {1/2/3 for adder, 1/2 for multiplier}

rs, rt: Source register numbers for an operation

Q_i, Q_k: The reservation stations which will provide input data for the operation

Q_i : The reservation station that contains the operation whose result should be stored into this register.