Notation

cf. Figure 6.18.

The pipeline registers keeping information between stages are labelled by the names of the stages they connect (e.g., IF/ID, ID/EX etc.).

Information flows from left to right except:

- Writing the result register – Potential for data hazard (dependence)
- Modifying the PC in case of a successful branch – Potential for control hazard

Tracing one instruction through the 5 stages – Stage 1

Stage 1: Instruction fetch and increment PC.

This step is identical for all instructions.

- Instruction is fetched from instruction memory. It is needed in following stages. So put it in IF/ID (as it was put in an IR for the multiple instruction cycle)
- PC is incremented. But since it will be incremented at every cycle by subsequent *instructions*, must be kept for computing target address in case we had a branch instruction.

Main resources needed: Instruction memory and 1 ALU.

IF/ID contains: Instruction and incremented PC

Tracing one instruction through the 5 stages – Stage 2

Stage 2. Instruction decode and register read.

Although at this point the opcode is known, let's save in register ID/EX everything that might still be needed in later stages.

- The incremented PC and the instruction
- The values of registers rs and rt (this is equivalent to what was stored in non-ISA visible registers A and B in the multiple cycle implementation)
- The 16 bits of the immediate instruction sign extended. This is needed if we have an immediate instruction, a load/store instruction (to compute the address) and a branch instruction (note that we do not compute the target address in this stage ;we could and we could store the result in ID/EX or even optimize further – see later)

Main resources needed: Register file

ID/EX contains: Instruction, incremented PC, the contents of the source registers, a 32-bit immediate value

Tracing one instruction through the 5 stages – Stage 3

Stage 3. This stage really depends on the type of instruction being executed. Assume a Load.

This stage will be "address computation". The ALU sources (rs and immediate) come from ID/EX. The result of the ALU will be stored in EX/EM.

Now, rs, rt, and the offset field are no longer needed (it does not seem that the PC is needed either but we keep it because of possible *exceptions* that we'll discuss later in the course). But we need to keep in EX/EM:

- The instruction (or at least the rt field to identify the register to write and the indication that we have a load) and the PC
- The sum of rs and the sign-extended offset field, i.e., the load address

Main resources needed: ALU (in fact if we compute the target address and the results of the compare, we need 2 ALU's)

EX/MEM contains: ALU result, name of the write register and indication from where that result will come from and the PC (more needs to be saved here, e.g., if we have a "store" instruction).

Tracing one instruction through the 5 stages – Stage 4

Stage 4 Still assuming a load. We need to read from memory and keep in MEM/WB the result of what we read.

For an arithmetic instruction, we would just forward relevant information (e.g., ALU result) from EX/MEM to MEM/WB.

Main resources needed: data memory

MEM/WB contains: result from the read (or from the ALU in previous stage), name of write register, and where the data to write comes from.

Tracing one instruction through the 5 stages – Stage 5

Stage 5 Write back. Nothing to be kept since last stage.

Summary of requirements for data path for pipeline (somewhat ideal)

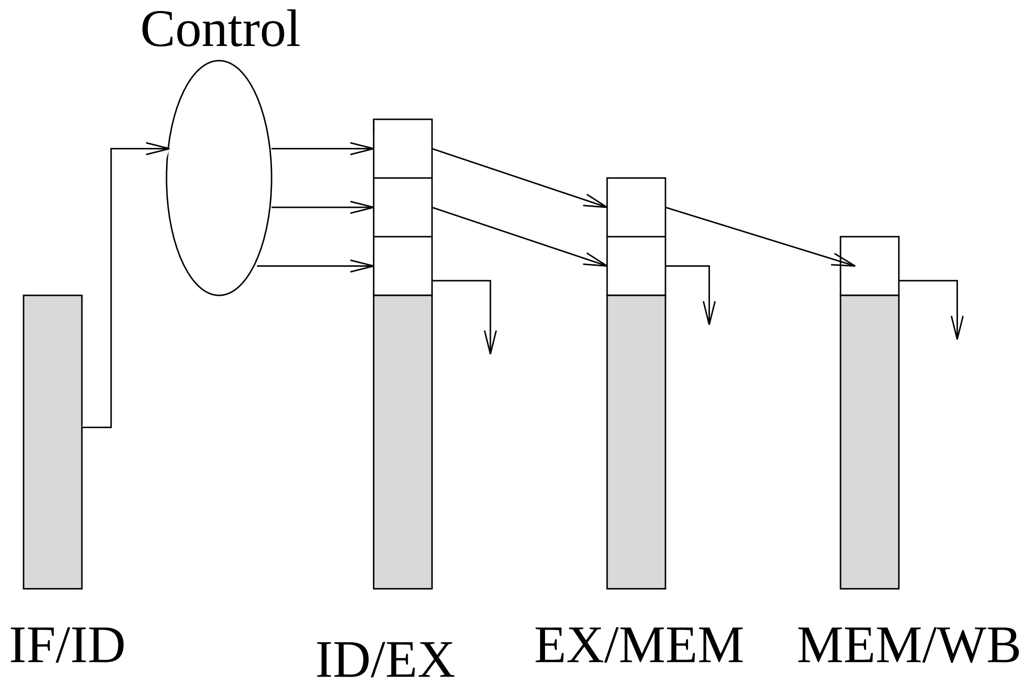
In stages 1 and 2 the information to be kept is the same for all instructions.

But the width of the pipeline registers must be accommodated to fit the "maximum" amount of information that needs to be kept.

For example, if we have a Store instruction, the contents of rs must be kept in EX/MEM (this was not the case for a Load or for an arithmetic instruction)

Instructions (except branches, see later) must pass through all stages even if nothing is done in one stage.

Control (highly abstracted)



Where are control lines needed

Pretty simple in the case of an ideal pipeline.

Cf. Figure 6.25

Control (Ideal case)

Control signals must be split among the 5 stages but there is no need for additional control (at least for the moment!)

- Stage 1: nothing special to control (read instruction and increment PC are asserted at each instruction)
- Stage 2: same thing every cycle. No special case.
- Stage 3: Control signals for ALUsrc and ALUop are needed. The control signal for Regdes (which register is to be written) has to be passed along to following stages.
- Stage 4: Controls for memory read or memory write and also for branches are asserted here.
- Stage 5: Control for what to write (memory output or ALUresult) and control to assert Regwrite are needed here.