Instruction-level Parallelism (Principles)

→ Pipelining

Generic task is converted into subtasks

→ RISC instruction set

Reduced Instruction Set Computer.

Properties:

- operations that affect the memory are the store and load.
- operations on data are arithmetic and logic that apply to data on registers.
- instructions formats are few in number.

↳ Non-pipelined implementation of RISC is.

1. **Instruction fetch** uses value of PC to get instruction from IR (instr. register.
2. **Instruction decode / register fetch cycle**: instruction is decoded, branch instruction is completed at the end of this cycle.
3. **Execution / effective address cycle**: ALU actions (memory reference, register-to-register or register-immediate).
4. **Memory Access**: load or store operation based on previous cycle. Store is completed at the end of this cycle.
5. **Write-back**: the result of register-to-register or load are written into registers of the

register - immediate

register bank.

→ Classical 5-stage pipeline for a RISC processor

Observations:

- Separate instruction and data memories, which means there is no conflict between IF and MEM stages.
- Register bank is accessed in two stages, the ID for reading and WB for writting. Register write is performed in the first half of the clock cycle and register in the second half.
- PC is updated every clock cycle in the IF stage. Potential branch target address is computed during the ID stage.

At the end of each stage, there are registers for temporary storage. At the beggining of every clock cycle, the results from a given stage are stored into registers that are used as inputs for the next stage.

→ Major hurdles of pipelining

3 classes of hazards:

- structural hazards: hardware conflicts in overlapped executions.
- data hazards: instruction depends on the results of the previous instruction
- control hazards: branch instructions and other instructions that affect the PC.

Hazards may require to stall the pipeline, in order to avoid the hazard. All instructions before the one that is stalled are also stalled, but the ones after are not. No new instruction is fetched.

→ Data hazards

Timing of instructions can result in data hazards (value calculated in one instruction is needed right after).

Solution? Forwarding (bypassing or short-circuiting)

It works as follows:

- ALU result from EX/MEM and MEM/WB are always fed back to the ALU inputs. The forwarding hardware takes care of choosing the correct input for the ALU (the forwarded value or register value)

- Also, some values may need to be forward from the data memory output into the data memory inputs.

Still, some data hazards can only be solved with stalls. One needs to add special hardware called pipeline interlock to preserve the execution pattern, which stalls the pipeline until the hazard disappears.

→ Control hazards

Predicted-untaken → treat every branch as untaken. If the branch is taken, the fetched instruction is turned into a no-op instruction and the [illegible] is restarted at the target address

fetch stage is

Predict taken → the branch target address is known before the branch outcome.

Delayed branch → the instruction right after the branch instruction is executed. The job of the compiler is to make the sequencial successor instruction valid and useful.

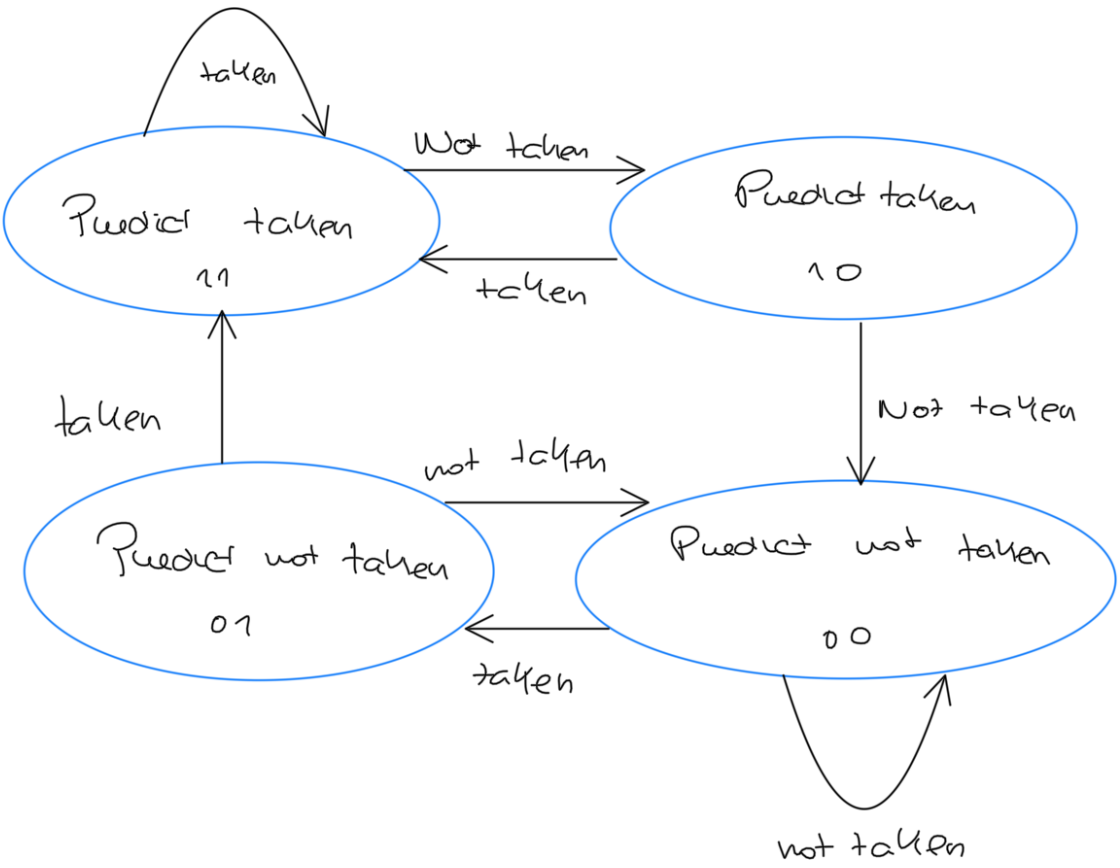
- from before: insert in that slot an instruction executed before the branch independent.
- from target: instruction that the branch points to (if there is no branch, one extra instruction is executed)
- from fall-through: instruction that will be executed next if the branch is not taken

As pipelines go deeper, delayed branches and other schemes are insufficient. 2 solutions:

- low-cost static schemes: rely on information available at compiler time, and have a bias towards taken or not taken.
- dynamic schemes: methods to predict during runtime.

The simplest dynamic branch-prediction uses a branch prediction buffer (BPF). It's a small memory indexed by the lower part of the address of the branch that states whether the branch was recently taken or not

2-bit prediction scheme state diagram.


taken
Predict taken
11
Not taken
Predict taken
10
taken
Not taken
taken
not taken
Predict not taken
01
Predict not taken
00
taken
not taken