Instruction-Level Parallel Processors

{**Objective**:
executing two or more instructions in parallel}

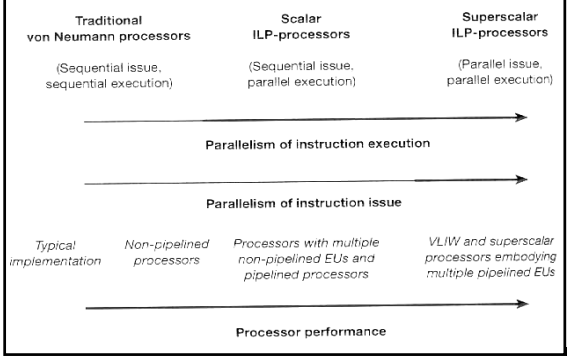
- 4.1 Evolution and overview of ILP-processors
- 4.2 Dependencies between instructions
- 4.3 Instruction scheduling
- 4.4 Preserving sequential consistency
- 4.5 The speed-up potential of ILP-processing

TECH
Computer Science

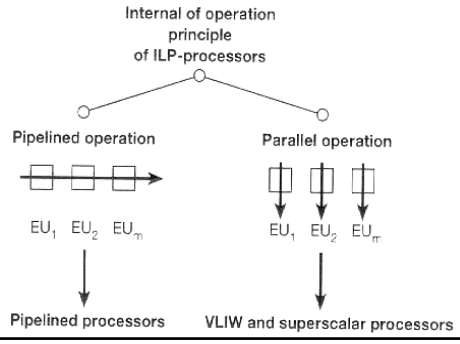
Improve CPU performance by

- increasing clock rates
  - → **(CPU running at 2GHz!)**
- increasing the number of instructions to be executed in parallel
  - → **(executing 6 instructions at the same time)**

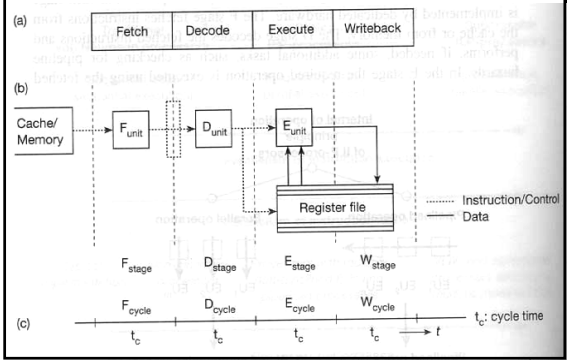
How do we increase the number of instructions to be executed?



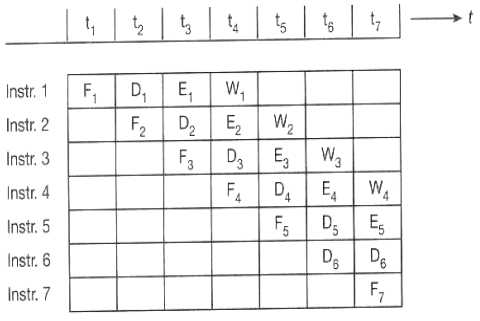
Time and Space parallelism



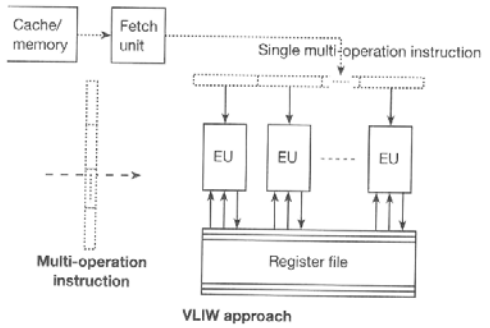
Pipeline (assembly line)



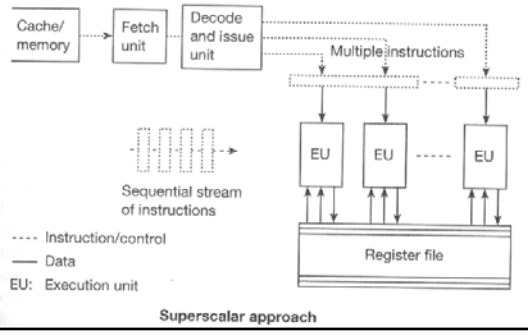
Result of pipeline (e.g.)



VLIW
(very long instruction word,1024 bits!)



Superscalar
(sequential stream of instructions)



From Sequential instructions to parallel execution

- Dependencies between instructions
- Instruction scheduling
- Preserving sequential consistency

4.2 Dependencies between instructions

- **Instructions often depend on each other in such a way that a particular instruction cannot be executed until a preceding instruction or even two or three preceding instructions have been executed.**
- 1 Data dependencies
- 2 Control dependencies
- 3 Resource dependencies

4.2.1 Data dependencies

- Read after Write
- Write after Read
- Write after Write
- Recurrences

Data dependencies in straight-line code (RAW)

- RAW dependencies
  - **i1: load r1, a**
  - **r2: add r2, r1, r1**
- flow dependencies
- true dependencies
- cannot be abandoned

Data dependencies in straight-line code (WAR)

- WAR dependencies
  - **i1: mul r1, r2, r3**
  - **r2: add r2, r4, r5**
- anti-dependencies
- false dependencies
- can be eliminated through register renaming
  - **i1: mul r1, r2, r3**
  - **r2: add r6, r4, r5**
  - **by using compiler or ILP-processor**

Data dependencies in straight-line code (WAW)

- WAW dependencies
  - **i1: mul r1, r2, r3**
  - **r2: add r1, r4, r5**
- output dependencies
- false dependencies
- can be eliminated through register renaming
  - **i1: mul r1, r2, r3**
  - **r2: add r6, r4, r5**
  - **by using compiler or ILP-processor**

Data dependencies in loops

for (int i=2; i<10; i++) {

x[i] = a*x[i-1] + b

}

- cannot be executed in parallel

Data dependency graphs

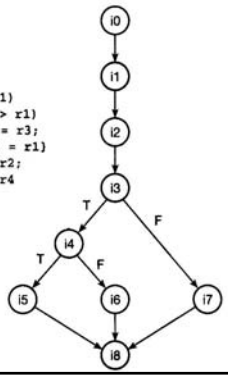
- i1: load r1, a
- i2: load r2, b
- i3: load r3, r1, r2
- i4: mul r1, r2, r4;
- i5: div r1, r2, r4

4.2.2 Control dependencies

- mul r1, r2, r3
- jz zproc
- :
- zproc: load r1, x
- :
- actual path of execution depends on the outcome of multiplication
- impose dependencies on the logical subsequent instructions

Control Dependency Graph

```
i0: r1 = op1;
i1: r2 = op2;
i2: r3 = op3;
i3: if (r2 > r1)
i4:     if (r3 > r1)
i5:         r4 = r3;
i6:     else r4 = r1)
i7: else r4 = r2;
i8: r5 = r4 * r4
```



Branches?
Frequency and branch distance

- Expected frequency of (all) branch
  - **general-purpose programs (non scientific):** 20-30%
  - **scientific programs:** 5-10%
- Expected frequency of conditional branch
  - **general-purpose programs:** 20%
  - **scientific programs:** 5-10%
- Expected branch distance (between two branch)
  - **general-purpose programs: every 3^{rd}-5^{th} instruction, on average, to be a conditional branch**
  - **scientific programs: every 10^{th}-20^{th} instruction, on average, to be a conditional branch**

Impact of Branch on instruction issue

- fig. 4.14

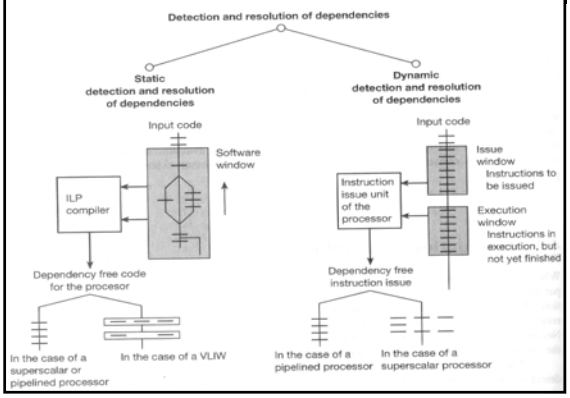
4.2.3 Resource dependencies

- An instruction is resource-dependent on a previously issued instruction if it requires a hardware resource which is still being used by a previously issued instruction.
- e.g.
  - **div r1, r2, r3**
  - **div r4, r2, r5**

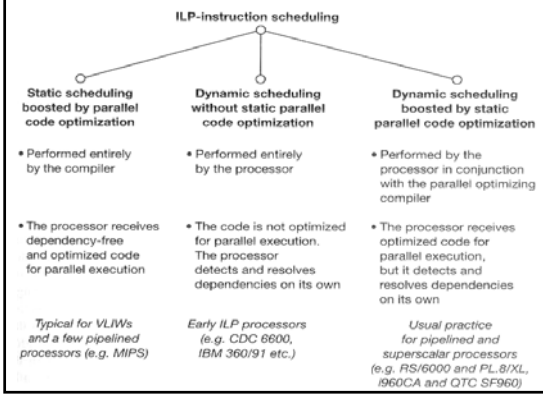
4.3 Instruction scheduling

- scheduling or arranging two or more instruction to be executed in parallel
  - **Need to detect code dependency (detection)**
  - **Need to remove false dependency (resolution)**
- a means to extract parallelism
  - **instruction-level parallelism which is implicit, is made explicit**
- Two basic approaches
  - **Static: done by compiler**
  - **Dynamic: done by processor**

Instruction Scheduling:



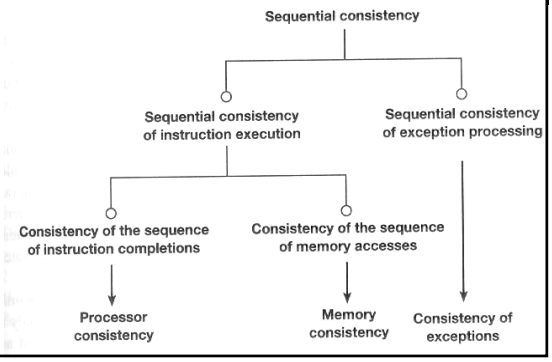
ILP-instruction scheduling



4.4 Preserving sequential consistency

- care must be taken to maintain the logical integrity of the program execution
- parallel execution mimics sequential execution as far as the logical integrity of program execution is concerned
- e.g.
  - **add r5, r6, r7**
  - **div r1, r2, r3**
  - **jz somewhere**

Concept sequential consistency



4.5 The speed-up potential of ILP-processing

- Parallel instruction execution may be restricted by data, control and resource dependencies.
- Potential speed-up when parallel instruction execution is restricted by true data and control dependencies:
  - **general purpose programs: about 2**
  - **scientific programs: about 2-4**
- Why are the speed-up figures so low?
  - **basic block (a low-efficiency method used to extract parallelism)**

Basic Block

- is a straight-line code sequence that can only be entered at the beginning and left at its end.
- i1: calc: add r3, r1, r2
- i2: sub r4, r1, r2
- i3: mul r4, r3, r4
- i4: jn negproc

- Basic block lengths of 3.5-6.3, with an overall average of **4.9**
  (RISC: general 7.8 and scientific 31.6)
- Conditional Branch → control dependencies

Two other methods for speed up

- Potential speed-up embodied in loops
  - **amount to a figure of** 50 **(on average speed up)**
    - unlimited resources (about 50 processors, and about 400 registers)
    - ideal schedule
- appropriate handling of control dependencies
  - **amount to** 10 to 100 **times speed up**
    - assuming perfect oracles that always pick the right path for conditional branch
  - **→ control dependencies are the real obstacle in utilizing instruction-level parallelism!**

What do we do without a perfect oracle?

- execute all possible paths in conditional branch
  - **there are 2^N paths for N conditional branches**
  - **pursuing an exponential increasing number of paths would be an unrealistic approach.**
- Make your Best Guess
  - **branch prediction**
  - **pursuing both possible paths but restrict the number of subsequent conditional branches**
  - **(more more CH. 8)**

How close real systems can come to the upper limits of speed-up?

- ambitious processor can expect to achieve speed-up figures of about
  - **4 for general purpose programs**
  - **10-20 for scientific programs**
- an ambitious processor:
  - **predicts conditional branches**
  - **has 256 integer and 256 FP register**
  - **eliminates false data dependencies through register renaming,**
  - **performs a perfect memory disambiguation**
  - **maintains a gliding instruction window of 64 items**