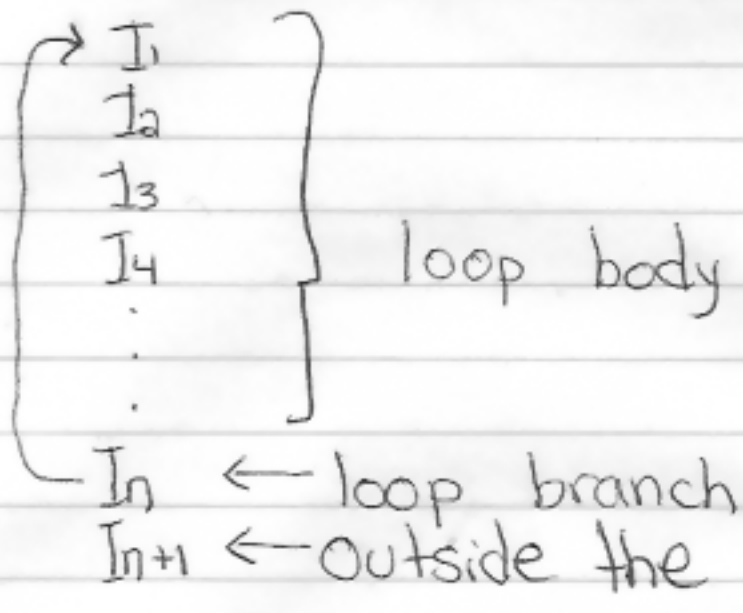
CS 3724 Jan 26/12

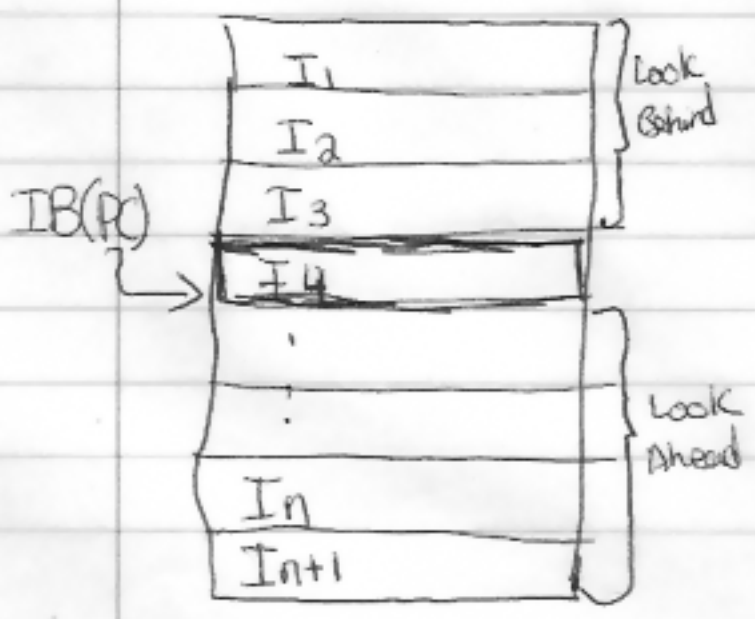
Look-Ahead Look-Behind Instruction Buffer
(How Program Structure Influences Computer Architecture)

Motivation: (two observations)
① most loops are fairly small
② for smaller loops, the branch to the top of the loop have bigger penalties

Idea:



Implementation of LALB Inst. Buffer:
① hold the entire loop inside the IB
② maintain the instruction in IB even after their execution (contrast w/ IBM 360/165)



IB(PC) [which is a mapping from PC] poinst to the current instruction

- Look-behind refers to inst already executed
- Look-ahead refers to the inst. to-be executed

When I_n (loop branch) is executed, either it goes back to the top of the loop (I_1) and hence it does not have to fetch those instructions ahead. Otherwise: ① I_{n+1} goes to the top of IB ② a "burst mode" read is issued to fetch a group of instructions to fill the IB.

Burst Mode Read - Reading a large group of information from consecutive locations

Some of the Assumptions are Realizable

- Multiplicity of Hardware is realizable
  - a) Several Registers (Buffers, Cache, ...)
    - Instruction Buffer
    - Operand Address Buffer
    - Operand Buffer
  - b) Multiple OpCode Decoders (DC)
  - c) Multiple Address Decoders (IA)
  - d) Multiple functional units (E-U)

Assumptions which are NOT Realistic

- IF, IU, DU times may not be the same
- All instructions do not take equal amounts of time in each one of the stage

Program Related Considerations

① Data dependency
② Order dependency
③ Control dependency
④ D-Unit dependency

} Among the inst in the program

(Not the same as inst. resource dependency)

Examples:

① I_1: ADD X, AC — AC ← AC+X
I_2: ADD Y, AC — AC ← AC+Y

a) Are I_2 and I_1 Data Dependent?
Yes
b) Are they order dependent?
No

② I_1: ADD X, AC — AC ← AC+X
I_2: MULT Y, AC — AC ← AC*y

a) Data Dependent?
Yes
b) Order Dependent?
Yes

Control Dependency

I_1; Branch to an interupt routine
I_2: ...

I_2 should not be executed until the interupt is processed

Important! In order to study the dependency issues in a broad context, we will use a computational (machine) model as follows:

We assume that

① Multi-functional
The machine has (m) functional units (D-Units) $D_1, D_2, \ldots, D_m$

② There are (n) general purpose data registers $R_1, R_2, \ldots, R_n$
These registers are global to all D-Units

③ Without any loss of generality, we assume that the machine has <u>two</u> types of instructions:
i) $R_i \leftarrow D_h (R_j, R_k)$
ii) If $P(R_i)$ then $PC \leftarrow L$
else sequential to the next inst
where P is a predicate
and L is the target address

④ Machine has instruction pre-fetch

⑤ Machine has operand pre-load
(ie. operands are in register $R_1, \ldots, R_n$)

Since the machine has instruction prefetch, we will not draw the S-F state.

*** I-U stage is responsible for issuing an instruction to a D-unit

An instruction can only be issued (to a D-Unit) IFF:

① The needed D-unit is available
② All the relevant registers are available
③ If an inst. can not be issued then none of the subsequent inst. can be issued.

ie Instruction issue is subsequent but issued inst can be executed in parallel

In-order issue, out-of-order execution