An Efficient Algorithm for Exploiting Multiple Arithmetic Units

Presented by Michelle
Outline

- Introduction to System/360 Model 91
- Definitions and Data Paths
- Designing Objectives and Solutions
- Conclusions
Divide the execution function into two independent parts
- Fixed-point execution area
- Floating-point execution area

Multiple execution units
- One adder
- One multiplier/divider
Definitions and Data Paths (I)

- Instruction unit — Floating-point operation stack (FLOS)
- Four floating-point registers (FLR)
- Buffer assigned to receive the storage operand — Floating-point buffer (FLB)
- Buffer assigned to store data — Store data buffer (SDB)
- Instruction format (except store and compare)

```
R1  op  R2  →  R1
Register  or  Register  Register
source  buffer source  sink
```
Definitions and Data Paths (II)
Objectives

- Achieve the greatest possible overlap of independent operations
- Preserve essential precedences
- Three functions
  - Recognize the existence of a dependency
  - Cause the correct sequencing of the dependent instructions
  - Allow independent instructions to proceed ahead
In order to preserve precedence, busy bit is used
- No instruction can be issued by the FLOS if the busy bit of its sink is on
- Not a maximum possible overlap of independent

Solution 1: Use different registers
- Overload on programmers or compilers
- It doesn’t work in the following example
Overlap of Independent Operations (II)

- The second add and the multiply should execute simultaneously
- Unfortunately, it doesn’t happen.
- The first add must wait for the result of multiply.
- The FLOS cannot decode an instruction unless a unit is available to execute it.
Solution 2: Associate more than one set of registers (control, sink, source) with each execution unit — reservation station

- Like there are multiple adders and multipliers/dividers
- In the Model 91, there are three add and two multiply/divide reservation stations.
Overlap of Independent Operations (IV)
Overlap of Independent Operations (V)

- **Solution**: Expand the busy bit into a counter
  - Appear to allow more than one instruction with a given sink to be issued.
  - As each is issued, FLOS increments the counter; as each is executed the counter is decremented.
  - Major difficulty is to keep the sequence.

LD F0, E
MD F0, D
AD F0, C
AD F0, B
AD F0, A

One problem hasn’t been solved!
What if multiple instructions with one sink?
Common data bus (CDB)
- Connect the registers, sink and source registers of all reservation stations, including the store data buffers but excluding the floating-point buffers
- CDB is fed by all units that can alter a register and feeds all units that can have a register as an operand.
- The control part of CDB enumerates the units which feed the CDB — tag.
- A tag is generated by the CDB priority controls to identify the unit whose result will next appear on the CDB.
Data Path including CDB and Reservation Stations

Figure 4 Data registers and transfer paths, including CDB and reservation stations.
How does CDB Work? (I)

- Adder 1 (A1) and Adder 2 (A2) with tag 1010 and 1011 respectively
- After issuing the first instruction
  - Control bits of F0: BB 1  TAG 1010 (A1)
- Issue the second instruction
  - Control bits of F0: BB 1  TAG 1011 (A2)
If CDB is free, A1 outgates its result and broadcasts the tag of the requestor (1010) to all reservation stations.

Each active reservation station compares its sink and source tags to the CDB tag.

If they match, the reservation station ingates the data from CDB.

CDB tag is also compared with the tag of each busy floating-point register in a similar way.
How does CDB Work? (III)

- AD2 cannot start until AD1 finishes because the source tag of AD2 is 1010 (A1)
- The result of AD1 cannot change register F0 once AD2 is issued because the tag of F0 is 1011 (A2)
Two Examples

LD F0, FLB1
DD F0, FLB2
STD F0, A
LD F0, FLB3
AD F0, FLB4

AD F0, FLB1
LDR F2, F0 move F0 to F2
- Tag of F0 is 1010
- Tag of F2 is 1010
- No unit or extra time required for LDR
Conclusions

- Reservation Stations
- CDB
- Preserve precedence while encouraging concurrency
Discussions

- Achieve local parallelism but not global parallelism — “looking ahead” about eight instructions
- Is there a good algorithm to achieve global parallelism?
- Is there a way to group the instructions smartly such that we can achieve best parallelism?