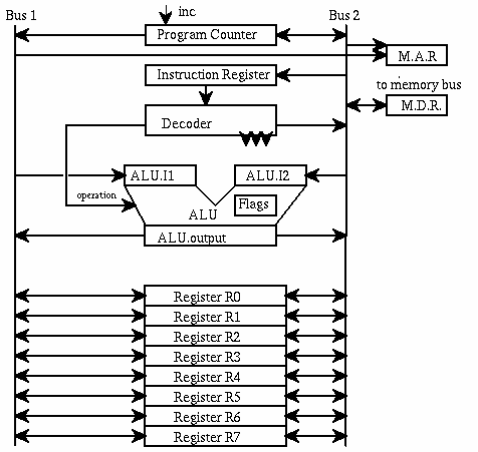
Parallellism inside the CPU

- The single-bus CPU operates in parallel only insofar as the data transfers are 1 word at a time
  - Otherwise the single bus forces transfers to be serialised.
- We can produce a 2-bus CPU:

With this architecture, we can have more than one transfer occurring at a time

- The actual register transfers required to implement an instruction are unchanged, but it may be possible to do more than one at a time.



Copyright 1998, 2002© Leslie S. Smith

What does parallelism buy us?

- Consider the add instruction:

  ADD.W R1, R2 ; add R2 to R1, leaving result in R1
- Nothing on instruction fetch

  MAR <- PC++

  MDR <- Memory[MAR]

  IR <- MDR
  - the problem there is different!
- But we can put both operands and the operation code into the ALU input registers simultaneously

  ALU.I1 <- R1 ALU.I2 <- R2 ALU.opn <- IR#BITS
- And nothing on storing the result

  R1 <- ALU.output
- So, the instruction takes 5, rather than 6 cycles.
- This is a bit disappointing.....

Copyright 1998, 2002© Leslie S. Smith

...and on a more complex instruction

- Consider again the (complex) instruction
  ADD.W R1, 123(R2)
- Again, instruction fetch is unchanged
  MAR <- PC++
  MDR <- Memory[MAR]
  IR <- MDR
- but we can organise the EA calculation better
  ALU.I1 <- R2 ALU.I2 <- IR#BITS ALU.operation <- IR#BITS
- although the operand fetch doesn't speed up
  MAR <- ALU.output
  MDR <- Memory[MAR]
- the actual calculation does speed up
  ALU.I1 <- MDR ALU.I2 <- R1 ALU.operation (unchanged)
  R1 <- ALU.output
- So there is some speedup: 8 cycles as against 10

Copyright 1998, 2002© Leslie S. Smith

Instruction Overlap

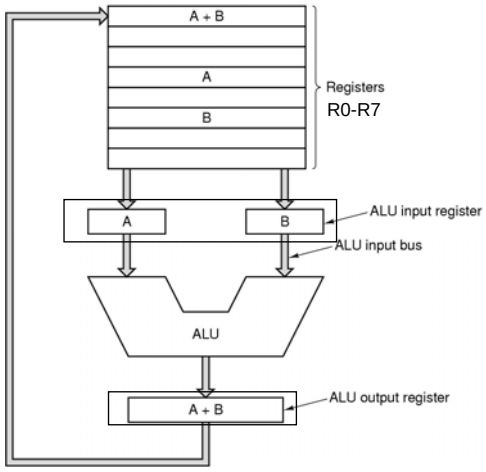
- Increasing parallelism inside the CPU allows
  instruction overlap - doing more than one instruction at a time
- Consider the pair of instructions
  LD.W R1, 1000
  ADD R1, R2
- We can arrange this as

MAR <- PC++	
MDR <- Memory[MAR]	
IR <- MDR	
MAR <- IR#BITS	
MDR<- memory[MAR]	
R1 <- MDR	MAR <- PC++
MDR <- memory[MAR]	
IR <- MDR	
ALU.I1 <- R1	ALU.I2 <- R2
R1 <- ALU.output	MAR <- PC++ (next instruction)

Copyright 1998, 2002© Leslie S. Smith

Parallelism Continued

- Datapath architecture
  - Most instructions consist of
  1. Moving data from registers to ALU inputs
  2. Performing ALU operations
  3. Storing the results
- With this architecture we can do all three at once (for different instructions)
- We can do better than simply overlapping instructions
  - Pipelining
- But before we come to that, we need to consider how we organise the actual operations that carry out instructions.



Copyright 1998, 2002© Leslie S. Smith

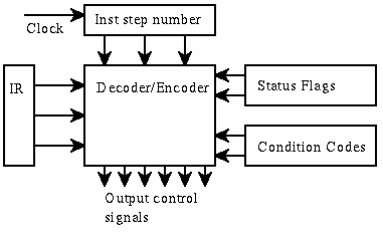
Sequencing Control Signals

- Major problem, particularly where we need to include parallelism
  - Because it makes matters a lot more complex
- There are two primary techniques
  - hardwired control
  - microprogrammed control
- Hardwired control
  - dedicated ('random') logic generate the control signals
- Microprogrammed control
  - a Microprogram sequencer generates the control signals
- But what exactly are the control signals required?
  - And what controls them

Copyright 1998, 2002© Leslie S. Smith

The CPU Control Unit

- The control Unit generates all the signals necessary to sequence the instruction execution
- Inputs
  - Instruction Register (IR)
  - System Clock
  - ALU condition registers
  - CPU status register
    - holds information about (e.g.) interrupt status, user/system mode etc.



- A simple instruction (e.g. LD.W) might not need all of these
  - but branches (control transfers) often do

Copyright 1998, 2002© Leslie S. Smith

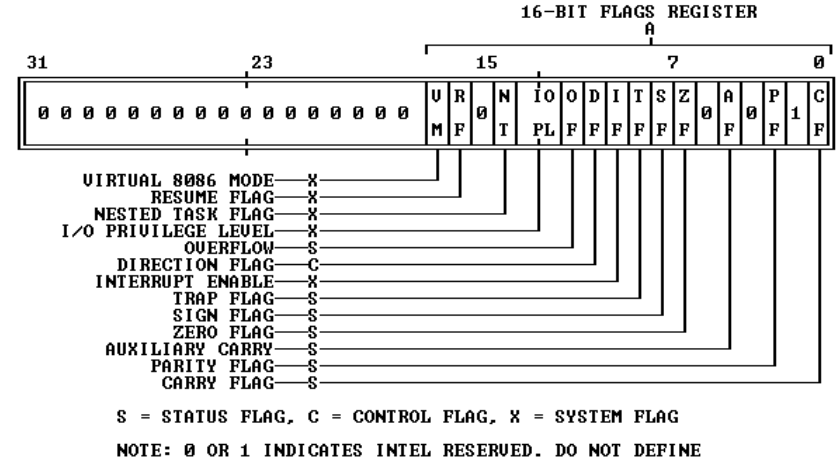
Condition Codes and Status Registers

- As well as the (long) registers in the CPU there are also 1-bit registers
  - sometimes called flag registers
- These hold information on what has recently happened at the CPU
  - often organised into a status word
    - processor status word
    - or condition register
- Some are set as a result of internal operations
  - these may be set or reset by operations that use the ALU, for example,
    - if the result of an ALU operation is zero,
    - if the result of an ALU operation is negative
    - and are often tested in branch instructions
  - or they may be set to distinguish system from user programs
  - or they may relate to interrupt state
- Some are set as a result of external factors
  - such as interrupts.

Copyright 1998, 2002© Leslie S. Smith

EFLAGS register on 80386.

Figure 2-8. EFLAGS Register



S = STATUS FLAG, C = CONTROL FLAG, X = SYSTEM FLAG

NOTE: 0 OR 1 INDICATES INTEL RESERVED. DO NOT DEFINE

Copyright 1998, 2002© Leslie S. Smith

Example: a branch instruction

- Consider the instruction

  JZ -24 % jump if zero
- I.e. load the PC from the location 24 back if the result of the last operation was 0
- After instruction fetch, the Z (zero) flag in the ALU is checked:
  - if Z != 1 the next instruction is loaded
  - if Z==1 the PC is decremented by 24
- Most CPUs provide a broad range of conditional branch instructions, and these use the ALU registers
- Additionally for nearly all instructions,
  - if the I (interrupt) flag is set, then operation continues at a different address
  - and, for many instructions, an exception is raised if the CPU mode is inappropriate
    - for example, user programs directly altering paging registers

Copyright 1998, 2002© Leslie S. Smith

Control Unit Signals

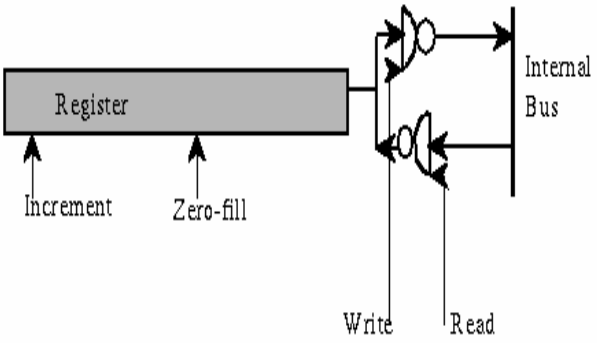
- The control unit is used to generate signals which
  - open/close data paths
    - e.g. links for reading or writing a register to/from a bus
  - increment registers
    - e.g. P.C.
  - set up ALU operations
    - i.e. send bit-strings to ALU operation register
  - zero-fill registers

Copyright 1998, 2002© Leslie S. Smith

Example:

register<-> bus connection

- Thick lines are datapaths
  - containing many wires (e.g.32)
- Thin lines are control lines
- Not all combinations of control lines are valid (e.g read and write)
- Note that connection is through NAND gate, so that register is negated on to the bus (and vice versa).
  - This does not matter (why?).



Copyright 1998, 2002© Leslie S. Smith

Control Lines for 1-bus CPU

- Decoders have a large number of outputs.
- For single-bus CPU
  - PC 3 lines (PC <-bus, bus<-PC, increment)
  - IR 2 lines (bus connection)
  - MAR 2 lines (MAR<-internal bus, external bus <- MAR)
  - MDR 4 lines (2 for internal bus, 2 for external bus)
  - ALU.I1, ALU.I2 1 line each
  - ALU.output 1line
  - ALU.operation 6 lines
    - 1 controlling ALU.operation <-IR
    - 5 defining actual operation
  - Registers 32 lines
    - each has 2 for bus (Ri<-bus, bus<-Ri), plus zeroise and increment
- Total: 52 lines.
- This excludes lines used
  - to access or zeroise the flag registers

Copyright 1998, 2002© Leslie S. Smith

Control Lines for the 2-bus CPU

- This decoder needs even more lines. For single-bus CPU
  - PC 4 lines (PC <-bus2, bus2<-PC, bus1<-PC, increment)
  - IR 2 lines (bus connection)
  - MAR 3 lines (MAR<- bus1,MAR<-bus2, external bus <- MAR)
  - MDR 4 lines (2 for internal bus, 2 for external bus)
  - ALU.I1, ALU.I2 1 line each
  - ALU.output 1 line
  - ALU.operation 6 lines
    - 1 controlling ALU.operation <-IR
    - 5 defining actual operation
  - Registers 40 lines
    - each has 5 for bus (Ri<-bus1, bus1<-Ri, Ri<bus2, bus2<-Ri), plus zeroise, increment
- Total: 62 lines.
- What about the data path architecture ? See tutorial.

Copyright 1998, 2002© Leslie S. Smith