Design and Implementation of a FPGA-based Pipelined Microcontroller

Rainer Bermbach, Martin Kupfer
University of Applied Sciences Braunschweig / Wolfenbüttel
Germany

Embedded World 2009, Nürnberg, 03.03.09

OVERVIEW

- INTRODUCTION
- PIPELINING
- STRUCTURE OF THE PIPELINE
- HAZARDS AND THEIR HANDLING
- THE IMPLEMENTED PIPELINE STAGES
- RESULTS
- CONCLUSION

INTRODUCTION

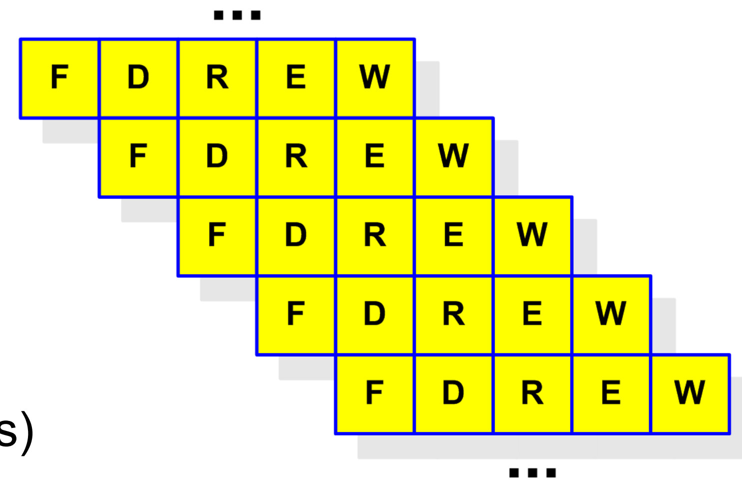
- PIC-compatible microcontroller (VHDL-PIC) had been developed
  - Compatible to a Microchip PIC 16c55x microcontroller featuring
    - 8-bit Harvard architecture
    - 14-bit instruction word (35 instructions)
    - hardware stack (return address only)
    - internal RAM space (up to 512 bytes organized in four banks)
    - program memory for up to 2k instructions
  - Developed for use in System-on-Chip (SoC) designs
  - Implemented on a Xilinx Spartan 3 FPGA
  - Clock frequency up to 100 MHz (approx. 20 - 22 assembler MIPS)
  - Enhancements like debug module, trace-buffer, statistic module

INTRODUCTION (continued)

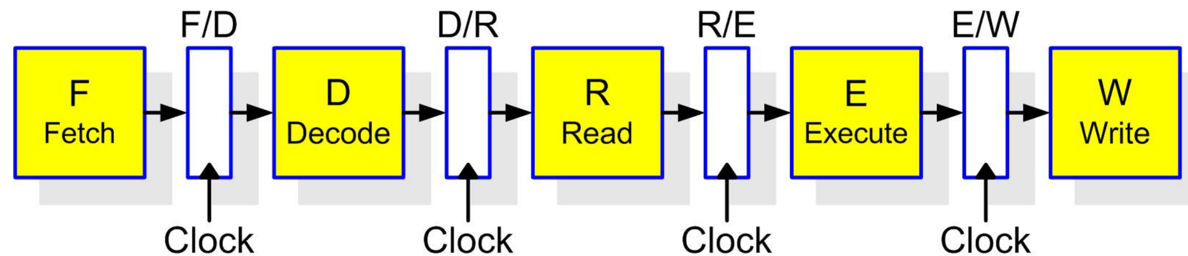
- Implementation of a multi-stage pipeline version of the VHDL-PIC
  - In principle, one instruction is executed every clock cycle
  - Five stages: FETCH, DECODE, READ, EXECUTE, WRITE back
  - Fighting problems due to non-sequential program flow
  - Mechanisms to prevent working with invalid data
- Motivation
  - Implementing a pipeline for increasing processing power
  - Study the actual problems in pipeline design, possible approaches and the concrete solutions
  - Better IP for System-on-Chip (SoC) designs
- Result: Nearly three times faster than standard VHDL-PIC

PIPELINING

- Goal: acceleration of processing speed
- Execution of one instruction is subdivided into subtasks, each executed in one clock cycle
- Instructions are processed in parallel in stages, so every clock cycle one instruction is concluded
- Hazards limit the processing speed:
  - Structural hazards (architecture)
  - Data hazards (data dependencies)
  - Control hazards (any kind of branches)



STRUCTURE OF THE PIPELINE



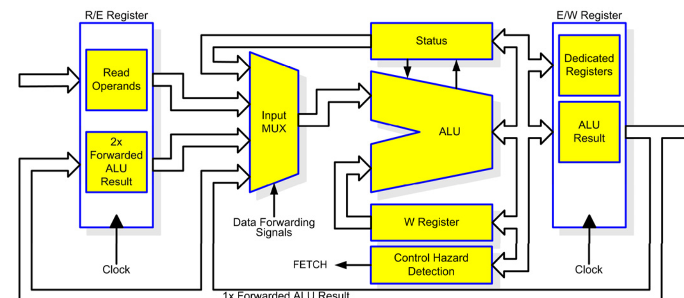
- Five processing stages:
  - FETCH: builds the program counter (PC) for the next instruction and fetches this instruction from the program memory
  - DECODE: analyzes the opcode and sets signals for following stages
  - READ: builds the address and reads the operand
  - EXECUTE: calculates the result of the instruction
  - WRITE back: writes the result back to respective memory position
- Pipeline registers: register between adjacent stages for passing information belonging to one instruction to the next stage.
  - Mechanisms for halting and flushing instructions

HAZARDS – Types of Hazards

- Structural hazards: non-exclusive access to hardware resources
  - e.g. program or data memory with one bus for read and write access does not allow simultaneous read and write action.
- Data hazards: two or more instructions in the pipeline need access to the same memory position
  - a memory position needs to be read while another instruction calculates its new value which is not written back yet.
  - writing to one register may effect the execution of other instructions (writing to status register, ports).
- Control hazards: a branch invalidates already fetched instructions and the pipeline has to be restarted with the jump's destination address.

HAZARDS – Structural / Data Hazard Handling

- Structural hazards: handled through use of additional hardware
  - All stages have exclusive access to necessary hardware resources e.g. dual-port RAM for memory, different busses for read and write access.
- Data hazards: detected and handled in READ on basis of addresses
  - Data hazard detection unit detects all possible data hazards using the addresses of write operations
  - Signals are set for different actions:
    - Data forwarding: the input of the EXECUTE stage can use new results instead of the data read
    - Halting the pipeline: waiting until data is written back (Status register and port access)



HAZARDS – Data Hazard Detection Unit

- An instruction writing a result to a memory position may release a data hazard, a so- called condition instruction.
- A read of the same or a depending memory position within a defined amount of cycles activates a data hazard, a so-called trigger instruction.
- The DHDU detects all condition instructions and sets signals for handling data hazards depending on the type, when a trigger instruction occurs
- All instructions are stored in a shift register for two cycles, another register tags a condition instruction

HAZARDS – Data Hazard Detection Unit (cont.)

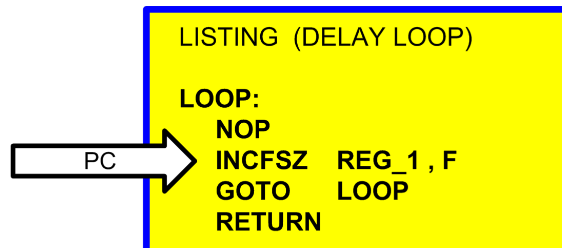
HAZARDS – Control Hazards

- Control hazards: detection as early as possible for fast reaction
  - Normal branches (destination included in the opcode) are detected in DECODE and the destination is fetched in the same clock cycle
    - To handle return and call instructions efficiently the stack is realized as dual-port RAM
      - One port accesses the stack output via a read stack pointer
      - The other port writes the stack by use of a write stack pointer
      - Therefore stack access is done in one cycle
  - Computed jumps are detected in READ (PCL as target address)
    - All instructions in the pipeline become invalid, the pipeline is halted until the destination address is calculated in EXECUTE
    - The same cycle the destination address is calculated the respective instruction can be fetched by forwarding the ALU result to FETCH
  - Conditional jumps are described on next slide

HAZARDS – Handling Conditional Jumps

- Conditional jumps (CJ) are assumed not to skip the subsequent instruction, decision is made in EXECUTE depending on the result – type of static branch prediction
- The CJ does not skip (Example 1)
  - The program execution is valid
- The CJ skips the subsequent instruction (Example 2)
  - The skipped instruction is no unconditional branch
    - The skipped instruction is flushed from the pipeline by filling the pipeline register between READ and EXECUTE with the value of a No Operation (NOP)
  - Skipping an unconditional branch (typically loop end, Example 3)
    - The branch is already executed in DECODE and the instructions now in DECODE and FETCH are invalid. Together with the one in READ they are removed from the pipeline and the next instruction is built in FETCH by using the PC_Branch register. ➔

HAZARDS – Conditional Jumps, Example 1



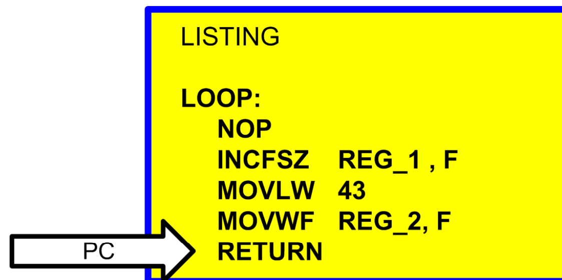
- The CJ does not skip
- The program execution is valid

Cycle	Register F/D	Register D/R	Register R/E	Register E/W
1	NOP	?	?	?
2	INCFSZ	NOP	?	?
3	GOTO	INCFSZ	NOP	
4	NOP	GOTO	INCFSZ	NOP
5	INCFSZ	NOP	GOTO	INCFSZ

…

- No cycle losses

HAZARDS – Conditional Jumps, Example 2



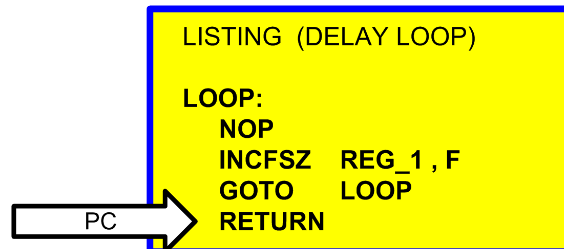
- The CJ skips the subsequent instruction
- The skipped instruction is no unconditional branch

Cycle	Register F/D	Register D/R	Register R/E	Register E/W
1	NOP	?	?	?
2	INCFSZ	NOP	?	?
3	MOVLW	INCFSZ	NOP	
4	MOVWF	MOVLW	INCFSZ	NOP
5	RETURN	MOVWF	NOP	INCFSZ

...

- The instruction to be skipped is flushed from the pipeline by filling register R/E with a NOP

HAZARDS – Conditional Jumps, Example 3



- The CJ skips the subsequent instruction
- The skipped instruction is an unconditional branch (loop end)

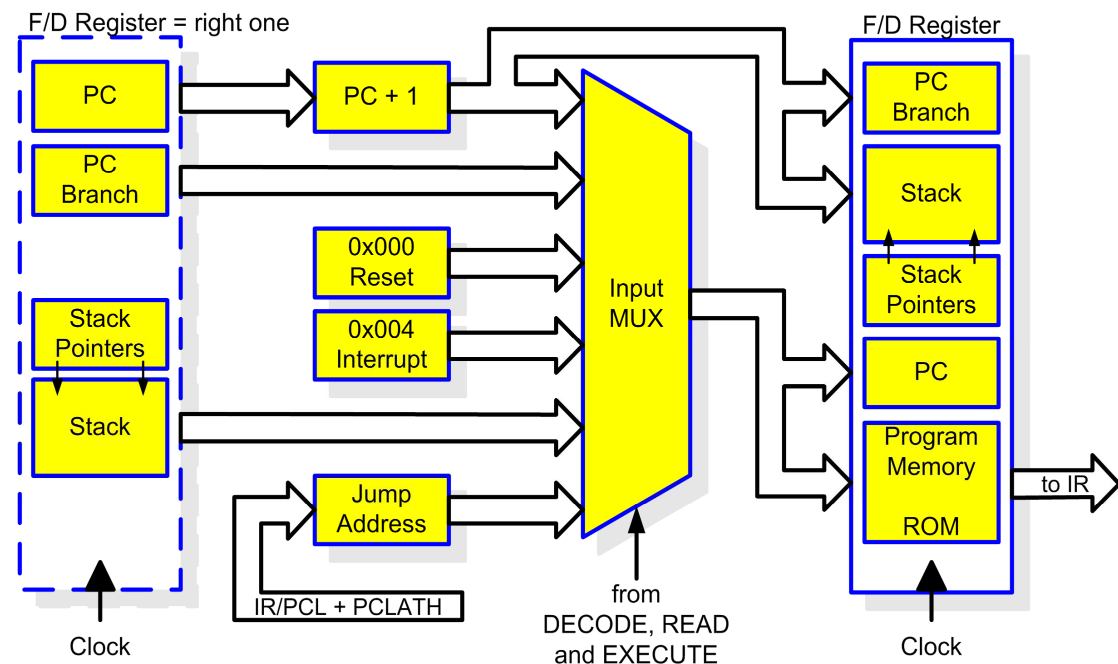
Cycle	Register F/D	Register D/R	Register R/E	Register E/W
1	NOP	?	?	?
2	INCFSZ	NOP	?	?
3	GOTO	INCFSZ	NOP	
4	NOP	GOTO	INCFSZ	NOP
5a	INCFSZ	NOP	GOTO	INCFSZ
5b	RETURN	NOP	NOP	INCFSZ

…

- The GOTO is already executed, instructions in FETCH, DECODE, READ are invalid → filled with NOP resp. newly built in FETCH by using the PC_Branch register

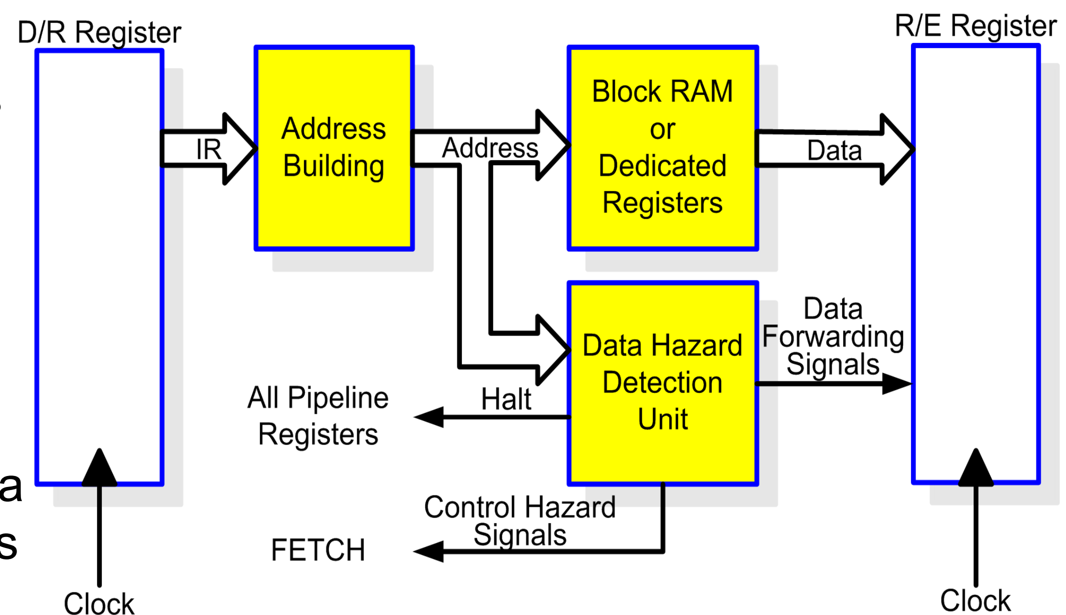
THE IMPLEMENTED PIPELINE STAGES

- FETCH
  - Creating the PC for the next instruction
  - PC is depending on several inputs from subsequent stages where control hazards may appear
  - The created PC is used to read the instruction code from the program memory
  - The PC is also written to a register to build a new PC in the next clock cycle



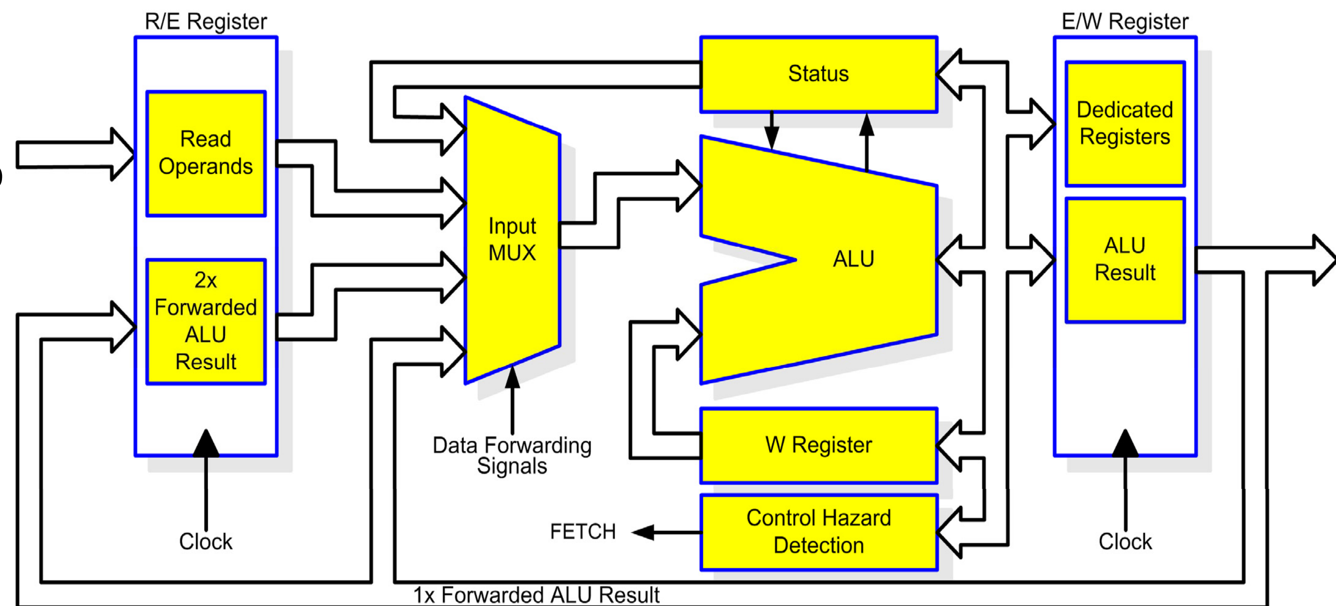
THE IMPLEMENTED PIPELINE STAGES (cont.)

- DECODE
  - In the DECODE stage the instruction code is analyzed and signals for subsequent stages are set
  - Control hazards are detected and signaled to FETCH
- READ
  - Building of the address for read and/or write access
  - Read the requested memory position from data memory
  - The DHDU detects data hazards and set signals for handling them



THE IMPLEMENTED PIPELINE STAGES (cont.)

- EXECUTE
  - Calculates the result of an instruction depending on signals from the previous stages
  - Signals computed jumps and skipping unconditional jumps to FETCH
  - Input depends on signals from READ (data hazard detection unit)
- WRITE back
  - Write back the result to respective memory position



RESULTS

- The pipelined VHDL-PIC was compared to a non-pipelined one using two different benchmarks (nearly best case and a worst case scenario = 71% conditional jumps with false branch prediction)

Program	Standard (cycles)	Pipelined (cycles)	Ratio	Speedup (x 70/100)
Embedded I/O	1,249,700	250,452	4.99	3.49
Floating Point	3,348	1,007	3.32	2.33

  – Standard (timing optimized) VHDL-PIC, up to 100 MHz with 25 assembler MIPS for sequential workload (typically 20 MIPS for normal workload depending on frequency of branches) (CPI = 4)
  – Pipelined PIC reaches up to 70 MHz resulting in 70 assembler MIPS for sequential workload (less for normal workloads) (CPI =1)
- Overall, the program execution is three times faster than in the standard version
- Not optimized in timing up to date

RESULTS (continued)

- The hardware needs for implementing the pipeline is analyzed by comparing the pipelined VHDL-PIC to the standard VHDL-PIC, too

Map report (total)	Pipelined	Standard	Increase in hardware
Number of slices (1,920)	772 (40%)	601 (31%)	+28%
Slice Flip Flops (3,840)	515 (13%)	356 (9%)	+45%
4 input LUTs (3,840)	1,097 (28%)	864 (22%)	+27%

- The amount of hardware is increased by approx. 30%, most of the additional hardware is used for
  - Data hazard detection unit
  - Input multiplexer (PC, ALU)
  - Pipeline register
- There are enough hardware resources left (in a Spartan-3 XC3S200) for implementing other System-on-Chip components

CONCLUSION

- Development and implementation of a multi-stage pipeline PIC compatible VHDL microcontroller
- Instruction execution is realized in five stages (F, D, R, E, W)
- Structural hazards are eliminated in concept phase by choosing dual-port RAM, separated busses …
- Data hazards are handled efficiently in the DHDU, data forwarding is used for eliminating any loss of cycles
- The pipeline is optimized for handling control hazards, most branches are executed without any loss of cycles, two cycles lost in worst case
- Up to three times faster at 70 MHz by only 30% increased hardware resources (14 times faster than standard PIC products at 20 MHz)

Thank you for your attention

I am now open for your questions

HAZARDS – Data Hazard Handling - Data Forwarding