

Computer Architecture Project

Requirements and Evaluation Criteria

Dr. Eng. Leszek Ciopiński

1. Project Objective

The objective of the project is to design and implement a microprocessor and its associated components in accordance with the specification and constraints provided in the project introduction and the requirements assigned to each team.

The processor must be implemented as a hardware design using Digital or Quartus II.

The project is divided into three stages. Each stage builds upon the results of the preceding stage. Students are expected to document their design decisions, explain the proposed solutions, and demonstrate the correct operation of the developed processor.

2. Stage I – Processor Architecture and Microinstruction Design (60 points)

Students must propose and document the architecture of the processor, including its instruction set, instruction encoding, internal data paths, interconnections between processor components, and microinstruction sequences implementing individual instructions.

The design must include:

- Processor architecture: the organization of the processor, its functional components, registers, buses, and other internal resources.
- Instruction set: the definition of the supported instructions, including their operation, operands, addressing modes, and effects on the processor state.
- Instruction encoding: the format and encoding of machine instructions, including the organization of instruction fields and the rules for decoding instructions. The instruction word must support variable-length encoding.
- Data path and interconnections: the organization of the data path and the connections between registers, the ALU, buses, memory interfaces, and other processor components, as applicable to the assigned project.
- Microinstruction design: the definition of microinstructions and the microinstruction sequences required to execute the supported machine instructions.
- Processor state and flags: the definition of the processor state, including the registers and status flags, and the rules governing their updates during instruction execution.

The proposed architecture and encoding must be consistent with the assigned project specification and all team-specific requirements.

3. Stage II – Control Unit Design (25 points)

Students must propose a design for the processor control unit responsible for coordinating the execution of instructions and controlling the operation of the data path.

The design must include:

- The organization and operating principles of the control unit.
- The method of decoding machine instructions and generating the required control signals.
- The method of sequencing microinstructions and controlling the execution of individual instructions.
- The cooperation between the control unit, the data path, and the instruction fetch unit.

- The handling of the processor state and status flags, where applicable.

For a hardware implementation, students must design the simplified control unit and instruction fetch unit.

The proposed control unit must be consistent with the architecture and microinstruction definitions established in Stage I.

4. Stage III – Complete Implementation and Testing (15 points)

Students must complete the processor design and demonstrate its correct operation.

Using Digital or Quartus II, this stage must include:

- Implementation of the complete control unit, instruction fetch unit, and memory management components, as required by the assigned specification.
- Integration of the processor components, including the data path, control unit, registers, and memory interfaces.
- Implementation of the required microprogram.
- Compilation of the complete processor design without errors in Digital or Quartus II.
- Preparation of a short test program that performs representative calculations and demonstrates the functionality of the implemented processor.
- Inclusion of the test program in the project files in MIF or HEX format, as appropriate for the selected tool and memory configuration.

The test program must demonstrate the correct execution of representative instructions and the interaction between the relevant processor components.

The project report must include the test program's pseudocode and assembly-language source code, together with an explanation of the expected results. The implementation must be tested, and the results must be presented in the report.

5. General Processor Requirements

The processor and its associated components must comply with the common requirements specified in the project introduction and the additional requirements assigned to each team.

Unless explicitly stated otherwise in the team-specific specification, the processor must:

- Support addressing of 4096 words of main memory.
- Support the default (implicit), immediate, and direct addressing modes.
- Support memory segmentation, with separate code and data segments.
- Include an appropriate number of segment registers. Include a program counter (PC) register that can be read but not directly modified by ordinary instructions.
- Update the status flags after the execution of the relevant instructions.
- Use a variable-length machine instruction format.

Any additional requirements concerning the number and types of registers, bus widths, supported instructions, addressing modes, memory organization, and other architectural parameters are defined in the team-specific project assignment.

6. Dependencies Between Stages and Assessment Rules

The project is assessed on a scale of 100 points:

- Stage I – Processor architecture and microinstruction design: 60 points.
- Stage II – Control unit design: 25 points.
- Stage III – Complete implementation and testing: 15 points.

Each stage depends on the successful completion of the preceding stage. Points for Stage II can be awarded only if the relevant Stage I design elements are correct. Similarly, points for Stage III can be awarded only if the relevant Stage II design elements are correct.

The maximum number of points for a stage is not awarded automatically upon submission. The work must meet the requirements specified for that stage.

The assessment considers the correctness, completeness, and consistency of the design, its compliance with the assigned specification, and the evidence provided to demonstrate that the implemented or simulated processor operates correctly.

7. Project Documentation and Submission

The submitted project must include the materials necessary to evaluate the design and verify its operation. These materials must include, as applicable:

- A description of the processor architecture and its components.
- The instruction set and machine instruction encoding.
- Data path diagrams and component interconnections.
- The microinstruction format and microinstruction sequences.
- The control unit design and instruction execution mechanism.
- The complete hardware design.
- The test program in pseudocode and assembly language, together with an explanation of its operation and expected results.
- The required project files, including the MIF or HEX file for hardware implementations where applicable.
- Evidence of successful compilation or execution and the results of the test program.

All implementation and documentation details must be consistent with the specification and the requirements assigned to the team.

8. Project is for two members team

9. There will be presentation and defense of the results of the project at least at the last classes.

Project topics:

Operations for all teams:

Adding REG, st8
Adding REG, REG

Subtract REG, st8
Subtract REG, REG

Move REG, REG
Move REG, MEM
Move MEM, REG

Operation with memory - all possible addressing modes (if applicable):

- Implied / Implicit addressing – The instruction implicitly knows the operand location (e.g., in an accumulator or stack) without specifying an explicit address.
- Immediate addressing – The operand itself is directly embedded within the instruction's operand field as a constant value.
- Direct / Absolute addressing – The instruction contains the exact memory address where the operand is stored.

- Indirect addressing – The instruction specifies a memory location that contains the pointer (address) to the actual operand.
- Register indirect addressing – The instruction specifies a register that holds the memory address of the operand.
- Base-offset / Base addressing – The operand address is calculated by adding a fixed offset (displacement) to a value stored in a base register.

Topic	General purpose registers	Specialised registers	Bus size	Instructions
1	8	0	8	CMP JE AND
2	6	2	16	OR JMP INC
3	4	4	8	DEC ADC NEG
4	10	0	8	XLAT [adr] NOT INC
5	8	8	8	CMP JNG CALL
6	10	2	8	OR MUL ROL
7	12	4	8	POP SBB STC
8	8	0	16	XCHG JMP INC
9	2	6	8	DIV NOT DEC
10	8	0	8	CALL RET AND

Descriptions of instructions (basing on 8086 processor):

Data Transfer Instructions

- **MOV** - Copy data from a source operand to a destination operand (register or memory).
- **PUSH** - Decrement the stack pointer by 2 and push a word from a register or memory onto the top of the stack.
- **POP** - Pop a word from the top of the stack into a register or memory location, then increment the stack pointer by 2.
- **XCHG** - Exchange the contents of two registers or a register and a memory location.
- **IN** - Read a byte or word from an input port into the accumulator (AL or AX).
- **OUT** - Write a byte or word from the accumulator (AL or AX) to an output port.
- **XLAT** - Translate a byte in AL using a lookup table pointed to by BX, loading the result back into AL.

Arithmetic Instructions

- **ADD** - Add the source operand to the destination operand and store the result in the destination.
- **ADC** - Add the source operand and the carry flag to the destination operand, storing the result in the destination.
- **INC** - Increment the specified register or memory location by 1.
- **SUB** - Subtract the source operand from the destination operand and store the result in the destination.
- **SBB** - Subtract the source operand and the carry flag from the destination operand, storing the result in the destination.
- **DEC** - Decrement the specified register or memory location by 1.
- **MUL** - Unsigned multiplication of the accumulator (AL or AX) by a source operand, storing the result in AX or DX:AX.
- **IMUL** - Signed multiplication of the accumulator by a source operand, storing the signed result in AX or DX:AX.
- **DIV** - Unsigned division of a double-length accumulator (DX:AX) by a source operand, storing the quotient and remainder.

- **IDIV** - Signed division of a double-length accumulator by a source operand, storing the signed quotient and remainder.
- **NEG** - Form the two's complement negation of the specified operand (change its sign).
- **CMP** - Compare two operands by performing a subtraction and setting flags without altering the destination operand.

Logical Instructions

- **AND** - Perform a bitwise logical AND operation between two operands, storing the result in the destination.
- **OR** - Perform a bitwise logical inclusive OR operation between two operands, storing the result in the destination.
- **XOR** - Perform a bitwise logical exclusive OR operation between two operands, storing the result in the destination.
- **NOT** - Perform a bitwise logical inversion (one's complement) of the specified operand.
- **TEST** - Perform a bitwise logical AND operation between two operands to update status flags without storing the result.

Shift and Rotate Instructions

- **SHL / SAL** - Shift the bits of an operand to the left by a specified count, filling lower bits with zeros.
- **SHR** - Shift the bits of an operand to the right by a specified count, filling upper bits with zeros (logical shift).
- **SAR** - Shift the bits of an operand to the right by a specified count, preserving the sign bit (arithmetic shift).
- **ROL** - Rotate the bits of an operand to the left by a specified count, wrapping bits around from MSB to LSB.
- **ROR** - Rotate the bits of an operand to the right by a specified count, wrapping bits around from LSB to MSB.
- **RCL** - Rotate the bits of an operand to the left through the carry flag by a specified count.
- **RCR** - Rotate the bits of an operand to the right through the carry flag by a specified count.

String Manipulation Instructions

- **MOVS / MOVSB / MOVSW** - Move a byte or word from the source string location (DS:SI) to the destination string location (ES:DI).
- **CMPS / CMPSB / CMPSW** - Compare bytes or words from two string memory locations and set flags accordingly.
- **SCAS / SCASB / SCASW** - Scan a string (in ES:DI) by comparing bytes or words against the value in AL or AX.
- **LODS / LODSB / LODSW** - Load a byte or word from string memory (DS:SI) into the accumulator (AL or AX).
- **STOS / STOSB / STOSW** - Store a byte or word from the accumulator (AL or AX) into string memory (ES:DI).

Control Transfer (Branching) Instructions

- **JMP** - Unconditionally jump execution to a specified target address or label.
- **JE / JZ** - Jump to the target address if the operands are equal or if the zero flag is set.
- **JNE / JNZ** - Jump to the target address if the operands are not equal or if the zero flag is clear.
- **JG / JNLE** - Jump to the target address if greater than (signed comparison).
- **JL / JNGE** - Jump to the target address if less than (signed comparison).
- **CALL** - Push the return address onto the stack and jump to a subroutine address.
- **RET** - Pop the return address from the stack and return control to the calling routine.
- **LOOP** - Decrement CX by 1 and jump to the target label if CX is not zero.

Processor Control and Flag Instructions

- **CLC** - Clear the carry flag to 0.
- **STC** - Set the carry flag to 1.
- **CLI** - Clear the interrupt enable flag, disabling external maskable interrupts.
- **STI** - Set the interrupt enable flag, enabling external maskable interrupts.
- **NOP** - No operation; wastes one clock cycle without performing any action.
- **HLT** - Halt the processor until an external reset or interrupt occurs.

