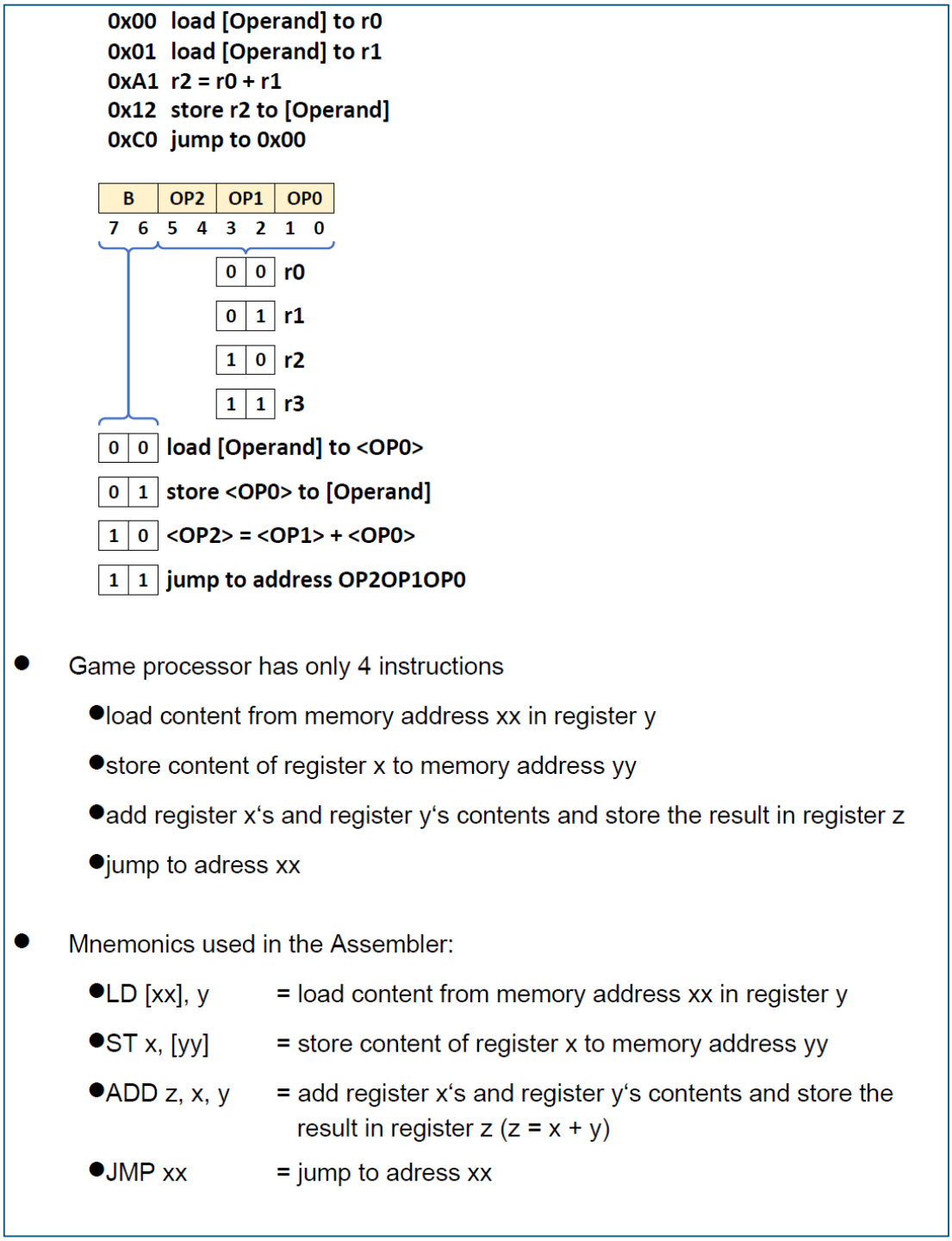
Preliminary requirement specification
Project „Small example processor emulation“
v0.2
Prof. Dr.-Ing. Stefan Widmann

In the Embedded Systems (1) lecture, a small example processor is used as an example of instruction encoding and decoding instead of a complex real-world processor. The small example processor shall be extended in the future. Its current encoding scheme is shown in the following figure:



- Game processor has only 4 instructions
  - load content from memory address xx in register y
  - store content of register x to memory address yy
  - add register x‘s and register y‘s contents and store the result in register z
  - jump to adress xx

- Mnemonics used in the Assembler:
  - LD [xx], y = load content from memory address xx in register y
  - ST x, [yy] = store content of register x to memory address yy
  - ADD z, x, y = add register x‘s and register y‘s contents and store the result in register z (z = x + y)
  - JMP xx = jump to adress xx

To help students learn about encoding and decoding, an emulator is to be created. It will allow students to better understand the functionality and basic operation of the processor.

The following features are required towards the emulation of the small example processor:

- The user shall be able to enter machine code
  - The emulator shall disassemble the machine code
- The user shall be able to enter assembler source code
  - The emulator shall assemble the source code to machine code
- The user shall be able to set input states (future extension of the processor)
  - The emulator shall always show the input states
- The emulator shall always show the output states (future extension of the processor)
- The emulator shall display all program memory contents
- The emulator shall show all data memory contents
  - The user shall be able to alter the data memory contents
- The emulator shall provide storing and loading memory contents (program and data memory)
- The emulator shall be able to decode and execute the program memory's contents
  - a step-by-step execution shall be possible
    - while stepping, the emulator shall highlight the currently executed program memory cell, and – if applicable – source and destination memory addresses in data memory
  - an animation mode shall be implemented, executing the program in a time step specified by the user; the highlighting described above shall be used
  - a run mode shall be implemented, executing the program without any intentional delay; no highlighting shall be used
  - the program counter's contents shall be shown by the emulator
  - the processor's status flag register contents shall be shown by the emulator

The following exercises taken from the lesson show typical problems to be solved by the students:

Problem 22:

Encode the following program for the game processor presented in the lecture:

- load [0x30] to r0
- load [0x31] to r1
- r3 = r1 + r0
- load [0x32] to r2
- r3 = r2 + r3
- store r3 to [0x33]
- jump to 0x00

What does the program calculate?

Solution

- instruction byte 0b00000000 = 0x00, followed by operand address 0x30, resulting in 0x00 0x30
- instruction byte 0b00000001 = 0x01, followed by operand address 0x31, resulting in 0x01 0x31
- instruction byte 0b10110100 = 0xB4, no additional operand

- instruction byte 0b00000010 = 0x02, followed by operand address 0x32, resulting in 0x02 0x32
- instruction byte 0b10111011 = 0xBB, no additional operand
- instruction byte 0b01000011 = 0x43, followed by operand address 0x33, also 0x43 0x33
- instruction byte 0b11000000 = 0xC0, no additional operand

The machine code of the program is therefore: 0x00 0x30 0x01 0x31 0xB4 0x02 0x32 0xBB 0x43 0x33 0xC0.

The program calculates the sum of the contents of the memory cells at addresses 0x30, 0x31 and 0x32 and stores the sum to the memory cell at address 0x33. The jump at the end of the program causes this calculation to be performed in an infinite loop.

Problem 23:

Decode the following instructions encoded for the game processor presented in the lecture:

23.1: 0xCA
23.2: 0xBF
23.3: 0x00 0x10
23.4: 0x43 0xF0
23.5: 0xFF
23.6: 0xDA

Solution

23.1: 0xCA : jump to address 0x0A
23.2: 0xBF : r3 = r3 + r3
23.3: 0x00 0x10 : load ⌊0x10⌋ to r0
23.4: 0x43 0xF0 : store r3 to ⌊0xF0⌋
23.5: 0xFF : jump to address 0x3F
23.6: 0xDA : jump to address 0x1A

Possible extensions to the requirements:

- allowing the future modification / extension of the encoding scheme by using e.g. a JSON or XML file to specify the layout of the instructions
    - the description scheme must be designed and documented in order to allow encoding map extensions and modifications