Extending Machine Instructions
How to turn a processor temporarily into a microprogram control unit
– Addendum –

Project history

The design ideas described here arose in the Eighties. The starting point was the task to develop a successor to a Z80-based multiprocessor system. Existing software had to be retained. So a transition to one of the then-contemporary 16-bit microprocessors was not feasible. A more detailed analysis had shown that only a few functions had to be accelerated. Extending the basic Z80 system by a control storage instead of designing a microprogrammed accelerator was a spontaneous idea.

License conditions

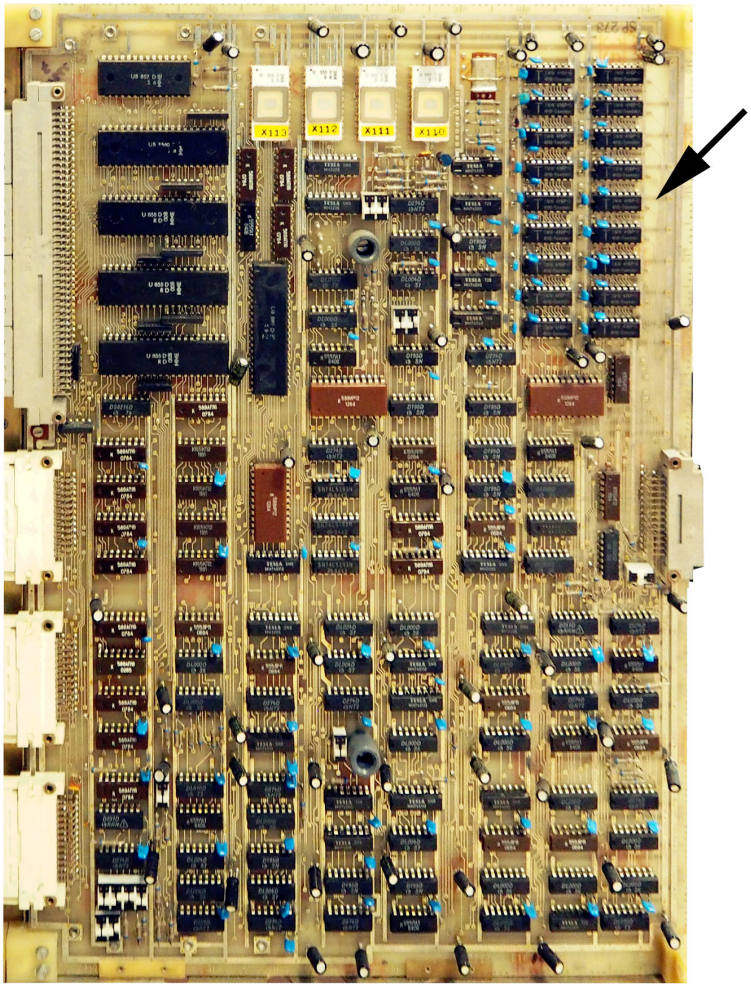

Figure A1 Here, a single-board computer (SBCs) is shown. It features an 9th memory bit, serving as parity or address compare stop bit (as shown in Figure 5). Details in [4] and [5]. The arrow points to the 32k • 9 bits DRAM memory, populated by 18 DRAMs of 16 kbits.

SBC Frame

```
A: 00  A': 00   IX: 00 00          TRACE CHAN: 0
F: 00  F': 00   IY: 00 00          STOP ON RD
B: 00  B': 00   SP: 00 00          STOP ON WR       ADRS 1: 00 00
C: 00  C': 00   PC: 00 00          CS ARMED         ADRS 2: 00 00
D: 00  D': 00   C  Z  S  P  H      PARITY           SET CS ADR
E: 00  E': 00   C' Z' S' P' H'     EXIT             CLR CS ADR
H: 00  H': 00   SBC REGS: 00 00 00                  SET CS BL
L: 00  L': 00   PART STATUS: 00 01 00 00 00         CLR CS BL
HDW ERR: SLV LOC RTOC ILAG PROVI PC ARBC DPP        RESTORE
```

Figure A2 The so-called SBC frame, displayed on a CRT. It allows for viewing and altering the content of the processor's registers, setting up compare stop modes, and single-stepping through the program. Menu items are selected via cursor keys (no mouse in those bygone times). A selected menu item is displayed inversely (dark characters in a white rectangle). In fields filled with zeros, hexadecimal numbers may be entered.

Extending a conventional microprocessor system

When reading instructions, the control storage is addressed in the same way as the conventional memory. The extension – principally an additional microinstruction – is loaded into the control storage data register (CSDR). The extension control circuitry energizes control signals to load data or addresses into output registers and feed register contents or literals to the data bus.

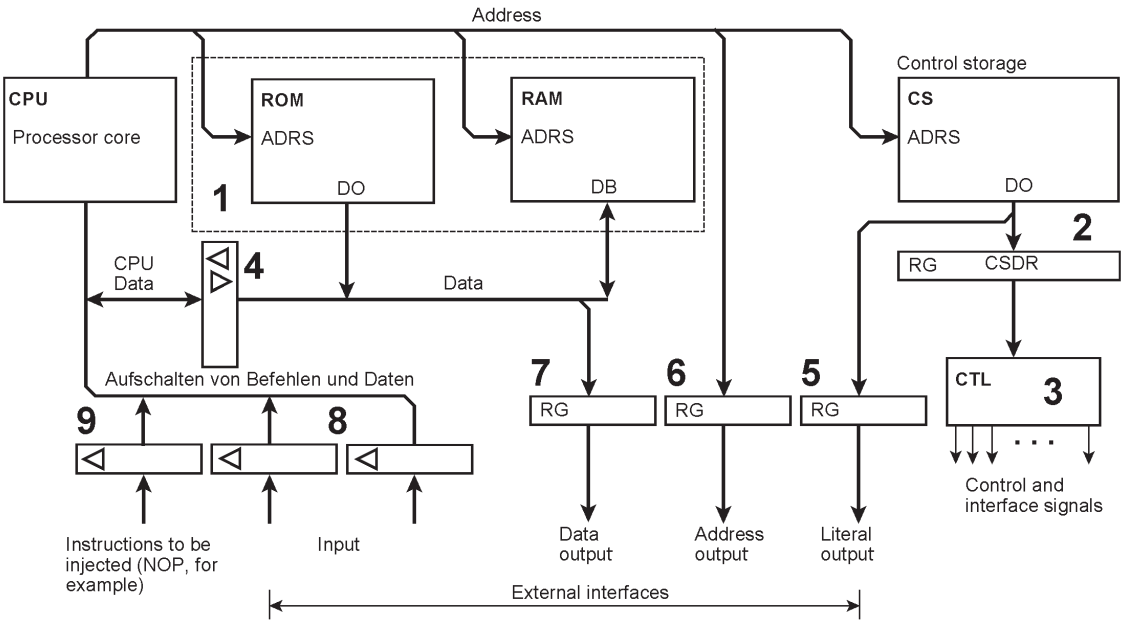

Figure A3 How the control storage and the accompanying circuitry fit into a traditional microprocessor system.

1 Conventional memories.
2 Control storage and control storage data register. Here the control storage is shown as a ROM. In practice, it is often a RAM that can be loaded in a particular access mode.
3 Extension control circuitry.
4 Bidirectional data bus buffer. Disconnects the conventional memorioes from the data bus when instruction or input data are to be injected.
5 Sideband output. The output data are literals in the microinstruction or come out of the extension control circuitry.
6 Address output. The data address of the current instruction is used as a bit pattern to be output.
7 Data output. The bit pattern on the data bus is output. It may come out of the processor or out of the memory.
8 Inject input data. If injected in a read cycle, they will be read by the processor, if injected in a write cycle, they will be stored.
9 Inject an instruction. An instruction from outside is fed to the processor instead of the instruction read out of the memory. A typical example is a NOP instruction causing the fetched instruction to be skipped.

Two examples of sideband effects

1. Disable interrupts temporarily

Sometimes there are sequences of instructions that must be not interrupted. A straightforward

example is the generation of a pulse by setting an output bit, waiting the required time, and finally clearing the bit, thus producing a pulse of a particular width. If this sequence is interrupted, the width of the pulse can increase unpredictably; a few hundred nanoseconds may become many milliseconds. The conventional remedy is to disable the interrupts by a DI instruction and enable them by an EI instruction after the pulse has been generated. More often than not, however, such program snippets are part of subroutines that run sometimes when interrupts are enabled and sometimes when disabled. In the latter case, the EI instruction will cause the program to crash, provided it runs often enough.

A bit position in the microinstructions can be used to disable the interrupts temporarily without impeding the interrupt control exerted by the interrupt enable flag (IF) in the processor's flag register.

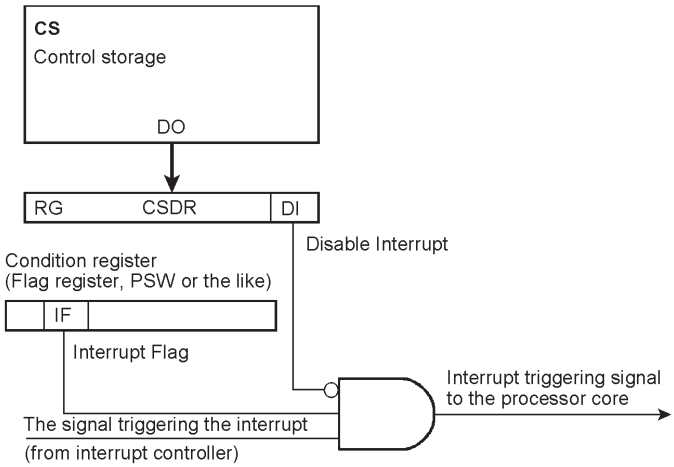

Figure A4 Disabling interrupts temporarily. It does not require additional instructions.

2. Load various registers

Conventionally, loading application-specific registers require appropriate I/O instructions. Here we solve this problem by accompanying microinstructions. Typical registers to be loaded this way are backup registers, capture registers, assembly registers (for outputs wider than a machine word), history buffers (for debugging and error handling), and the like. The advantage of this principle is that loading such registers does not require additional clock cycles and hence does not affect the real-time behavior of the machine.

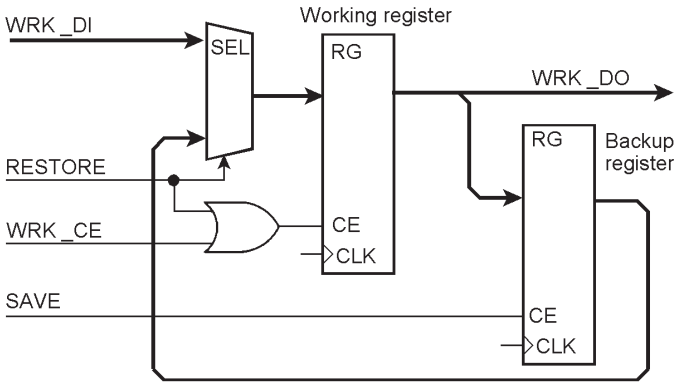

Figure A5 The content of a working register is to be saved into a backup register by a SAVE microcommand and restored by a RESTORE microcommand. WRK_DI/DO symbolize the input and output signals of the register flip-flops by which the working register is connected to the ambient circuitry.

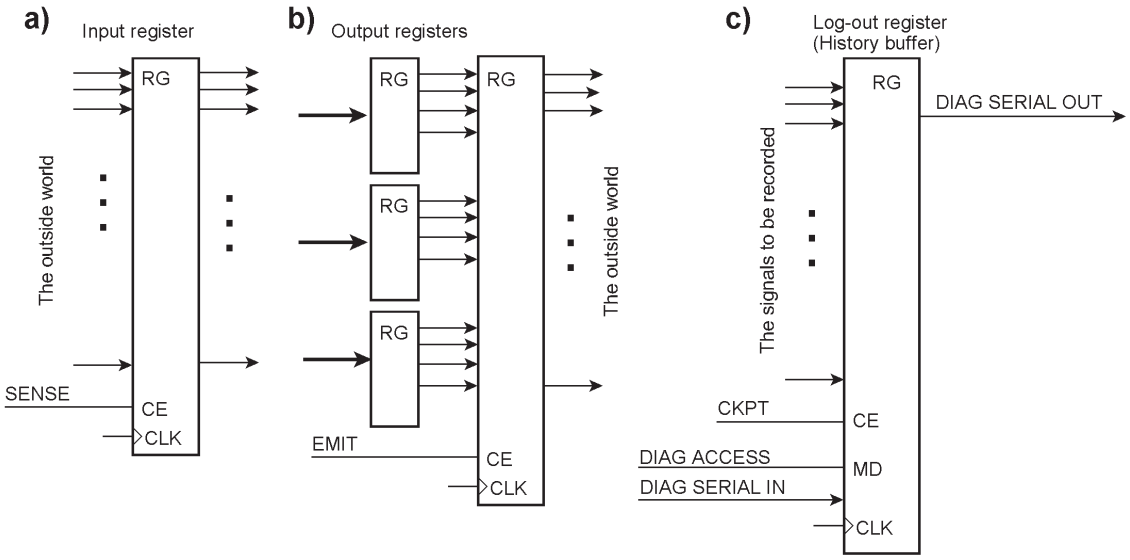

Figure A6 Various registers are loaded by microcommands. SENSE captures data from the outside world (a). EMIT causes the contents of various output registers to appear in the outside world at once (b). CKPT (Checkpoint) loads signals to be saved for debugging or error-handling into a log-out register or history buffer, respectively (c). In this example, it is read out serially.

Injecting instructions

In Figure 13 of the article, we have shown the idea of feeding the processor with a NOP instruction instead of the instruction it has addressed in the memory. This principle could be applied beyond the conditional execution of the instruction. The basic idea is to inject something other during the instruction fetch phase. When we inject NOPs, the instructions read out of the memory could be tapped for other purposes. In memory locations accompanied by appropriate extensions, arbitrary content could be stored. The stored words could be special-purpose instructions controlling an accelerator or merely immediate data to be output.

Thus it is possible to use the access width of the program memory and the instruction fetch cycles for output purposes. Consecutive instruction fetches – without data accesses in between – are, concerning data rate, often by far superior to programmed output loops.

Instead of NOPs, individual bits, bit fields, or complete instructions could be injected. Today, it is doubtless not appropriate to implement application-specific circuits this way. Instead of beefing up a processor with such tricks, we will simply choose a more powerful model.

A sometimes useful application, however, could be to extend the processor's instruction set by an EXECUTE instruction. Such an instruction causes a memory or register content to be executed as an instruction. If this instruction causes a branch, the program continues in the direction of the branch. Otherwise, the instruction following the EXECUTE instruction is executed. EXECUTE may be thought of as a subroutine consisting of only one instruction.

In the early days of computer development, it was common practice to modify instructions in the application program or to create them on the fly. For some time, however, so-called pure procedures are preferred. These are programs that may not be changed during execution. Here, the EXECUTE instruction is some kind of backdoor. This way, you may create your own instructions even in pure procedures. Sometimes, this may come in handy to speed up program sequences or circumvent shortcomings of the architecture.

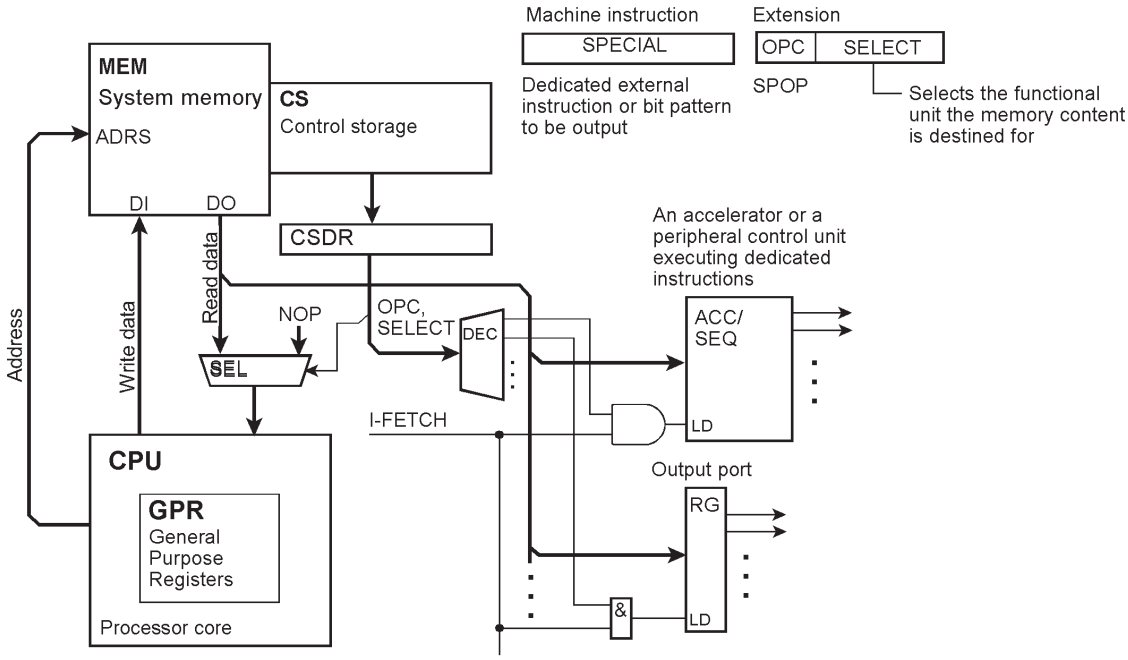

Figure A7 Instruction output. Instructions extended this way never make it into the processor. It will receive NOPs instead. So the memory may contain arbitrary bit patterns. They may serve as special instructions (a) of an accelerator or a peripheral control unit or may be output immediately (b).

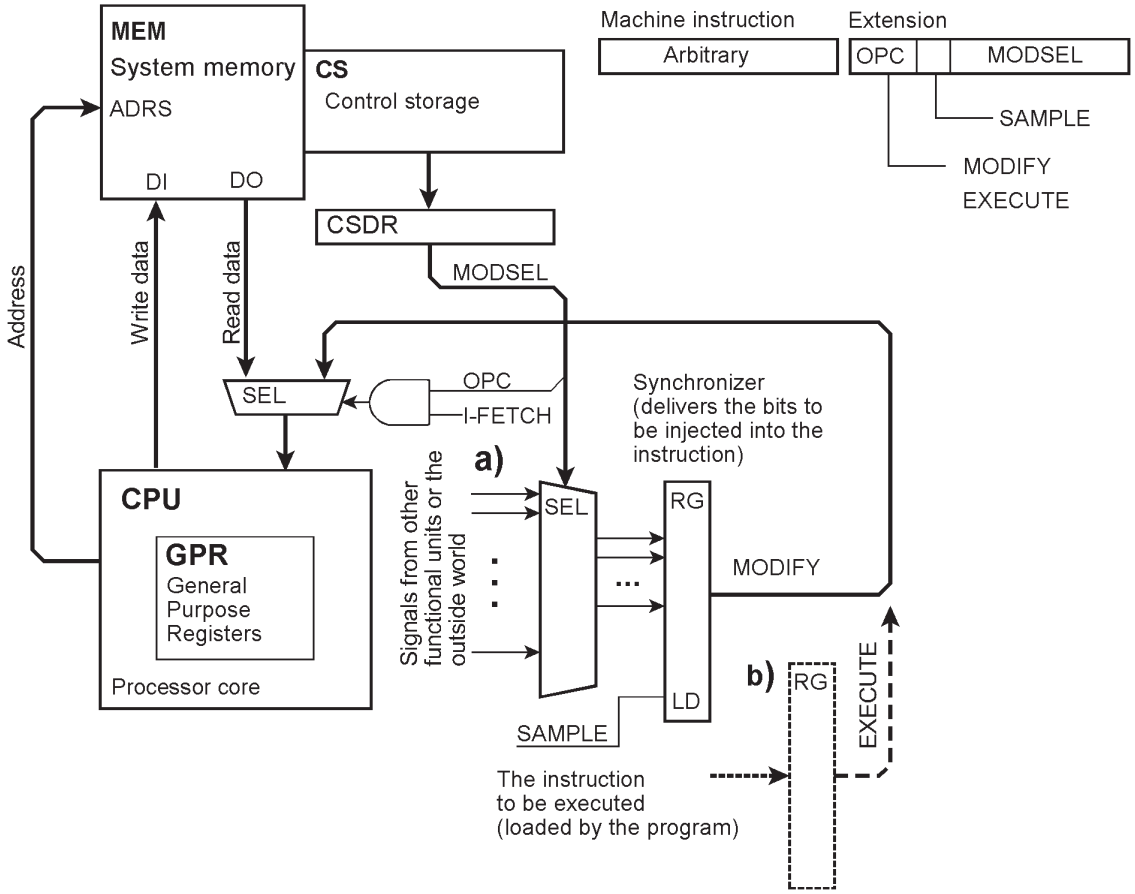

Figure A8 Modifying or substituting instructions from outside. Bits or bit fields of the instruction may be injected (a). Think, for example, of an address field or a literal value set or modified according to external conditions. Complete instructions could also be injected (b). Here a program-accessible register is shown to be loaded with an instruction that has been assembled by the application program. Injecting such an instruction is equivalent to the EXECUTE instruction provided in some legacy architectures. We could also think of supplying a complete instruction from outside, for example, from another processor in a multiprocessor system.

References

[1] Matthes, Wolfgang: Microprogramming Choices Explained (Part 1). Circuit Cellar, Issue 378, January 2022, p. 26-35.

[2] Matthes, Wolfgang: Microprogramming Choices Explained (Part 2). Circuit Cellar, Issue 379, February 2022, p. 22-32.

[3] Matthes, Wolfgang: Mikroprogrammierung. Prinzipien, Architekturen, Maschinen. ISBN 978-3-8325-5234-3. Logos, 2021.

German patent applications:

[4] Mikrorechneranordnung, vorzugsweise für den Einsatz in Multimikrorechnersystemen.
DE file number: DD 159 916 A1
Application number: 23096181
Application date: June 22, 1981
Microprocessor configuration, preferably for the application in multimicroprocessor systems.
EP000000067982A2

[5] Speicheranordnung mit Fehlererkennungs- und Diagnoseeigenschaften, vorzugsweise für Mikrorechner
DE file number: DD 225 072 6
Application date: Nov 10, 1980
https://register.dpma.de/DPMAregister/pat/register?AKZ=DD154244

[6] Speicheranordnung mit Eingabe-/Ausgabeanschluß, vorzugsweise zum Einsatz in Multimikrorechnersystemen
DE file number: DD 272 021 6
Application date: Dec 28, 1984
https://register.dpma.de/DPMAregister/pat/register?AKZ=DD233435

[7] Mikrorechneranordnung mit erweiterten Steuerwirkungen
DE file number: DD 288 148 1
Application date: Mar 21, 1986
https://register.dpma.de/DPMAregister/pat/register?AKZ=DD246858

[8] Mikrorechneranordnung mit programmgesteuertem Interfaceanschluß
DE file number: DD 288 145 7
Application date: Mar 21, 1986
https://register.dpma.de/DPMAregister/pat/register?AKZ=DD246860

All patents lapsed long ago.

Attaching accelerators:

[9] Patel, Sanjay; Hwu, Wen-mei: Accelerator Architectures. IEEE Micro, July-August 2008, p. 4-12.

[10] MicroBlaze Processor Reference Guide UG 081. Xilinx, 2009.

[11] Rosinger, Hans-Peter: Connecting Customized IP to the MicroBlaze Soft Processor Using the Fast Simplex Link (FSL) Channel. Application Note XAPP529. Xilinx, 2004.

[12] Madinger, Noah: The Co-Processor Architecture: An Embedded System Architecture for Rapid Prototyping. DigiKey, 2022.
https://www.digikey.com/en/articles/the-co-processor-architecture-an-embedded-system-architecture-for-rapid-prototyping

Sources

The author's project homepages:

https://www.realcomputerprojects.dev
https://www.controllersandpcs.de/projects.htm