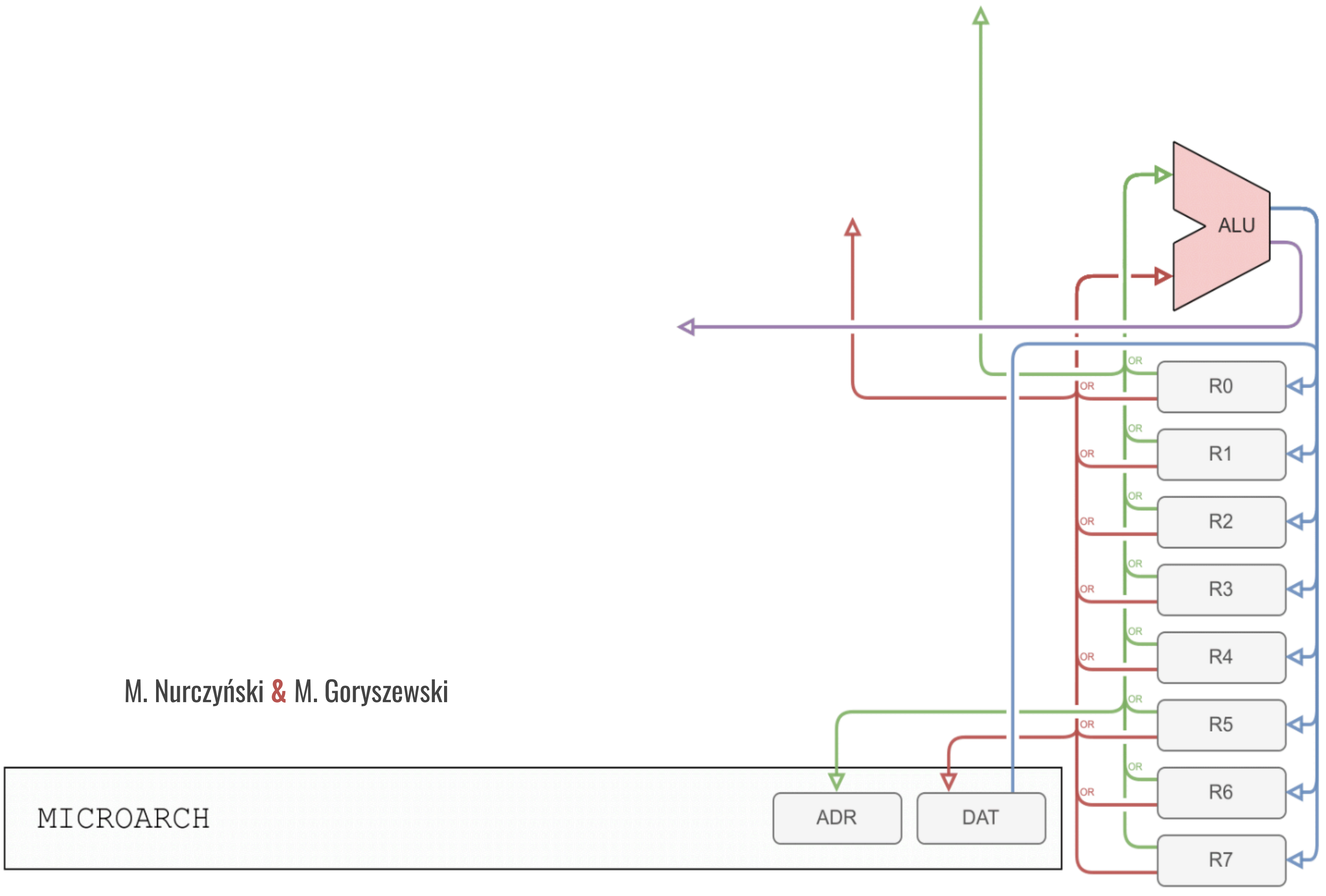
microarch

Instruction Set Architecture



M. Nurczyński & M. Goryszewski

Table of Contents

Overview

microarch is a minimalistic, 8-bit, extensible, RISC, Harvard, load-store computer architecture intended for small low-power (or power constrained) microcontrollers.

This document describes the assembly instructructions available to **microarch** programmers - their function, syntax, and encoding. Each instruction will be given its own section in this document. Some commonly expected information may be missing from this specification as it was designed to be implementation agnostic, as such it only defines the requirements and certain limits that all implementations must adhere to. To learn about those omitted here specifications consult the manual of your specific implementation.

---

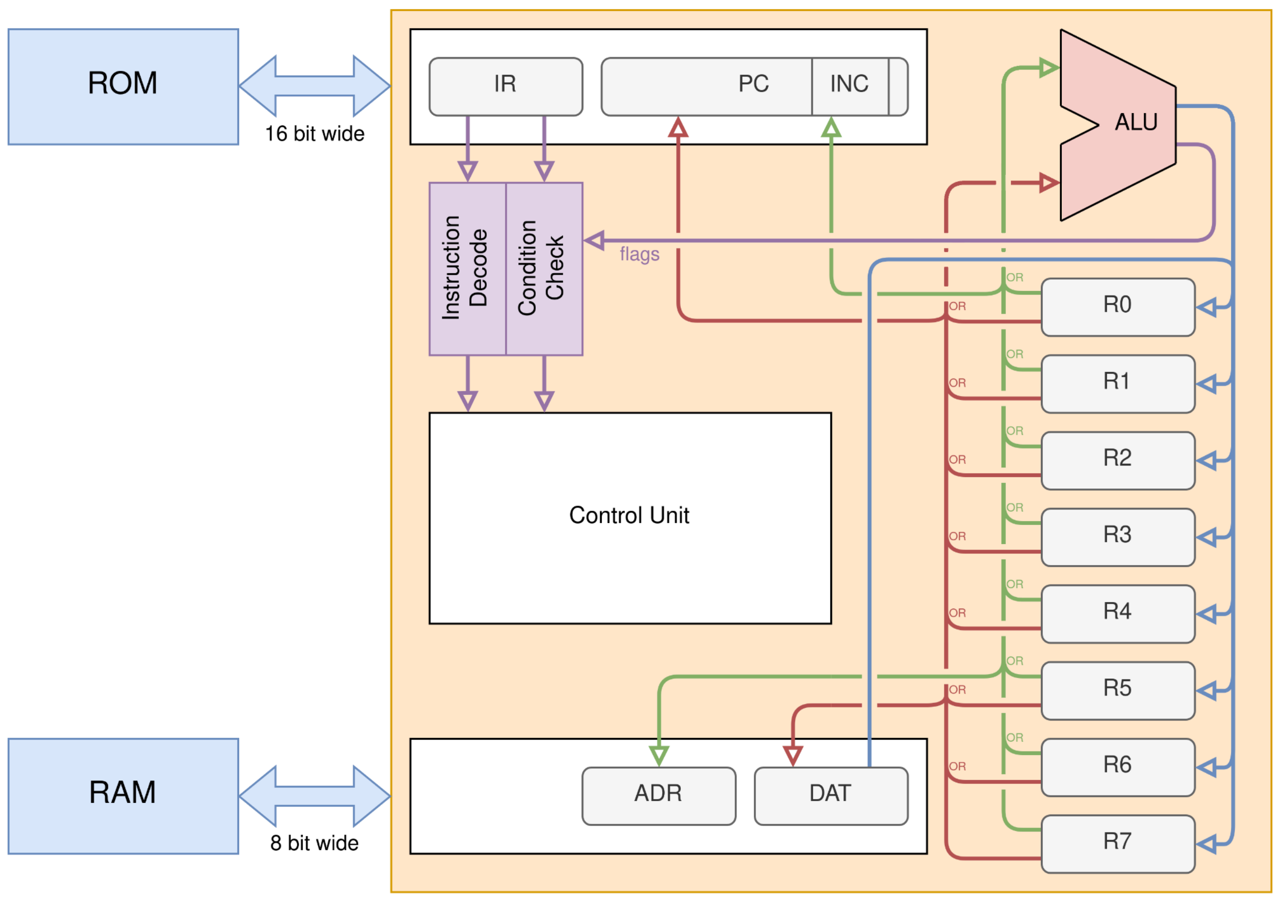
microarch offers 8-bit data address space and up to 16-bit instruction address space, allowing for up to 256 bytes of directly addressable data memory and up to 64kB of directly

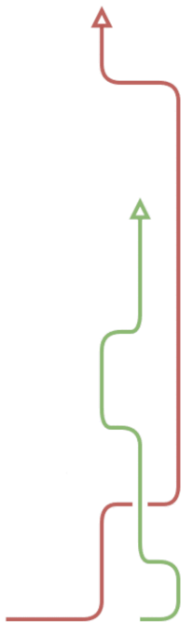
The microarch processor has 8 registers available to the programmer, numbered R0 through R7, with a powerful, unique register addressing system, that will be described later on in this document.

The architecture offers several power-saving stand-by modes that can be directly invoked by the program. It also offers a single interrupt line.

Block Diagram

Basic reference block diagram of a **microarch** core





Register Bank

There are 8 registers available to the programmer (numbered from R0 to R7), with a powerful, unique addressing system.

We recommend using the R7 register as stack pointer, in programs making use of stack (since there is no hardware support for stack in **microarch**, this is not enforced in any way).

Registers in **microarch** processors are addressed through special 8 bit values, described below, that will be referred to as "registry sets".

Registry sets

"registry set" is an 8-bit value (a byte) describing any selection of registers in the form of a bitmask:

R7	R6	R5	R4	R3	R2	R1	R0

The youngest bit references the R0 register, the second-youngest bit references the R1 register and the oldest bit references the R7 register. For example:

Byte 00001000 means register R4.
Byte 01101100 means registers R6, R5, R3, R2.
Byte 00000000 means no registers.
Byte 11111111 means registers R7, R6, R5, R4, R3, R2, R1, R0 (so all registers).

Internal Registers

Those registers are inaccessible to the programmer by their names, formally not defined by the ISA, and are listed here for context only.

Name	Size	Description
PC	16 bits	Program counter - pointer into the read-only program memory
IR	24 bits	Contains the instruction currently being executed
ADR	8 bits	Pointer into the read-write data memory for the current operation
DAT	8 bits	Temporary buffer for values read and written to and from memory

ACC	8 bit	Temporary buffer for ALU results before they are written back

Input registry sets

When a registry set is given as instruction input, the value used will be the result of a bitwise OR operation of all registers specified in the registry set.

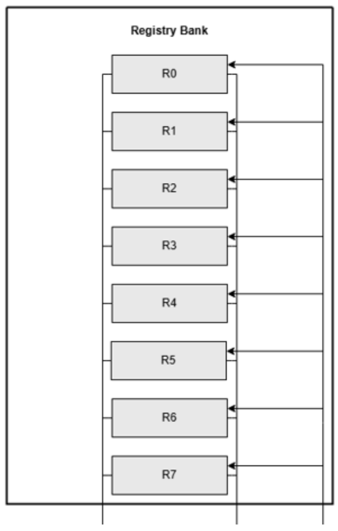
For example:
When R1==01110010 and R4==00110100, then the value of the registry set 00010010 will be 01110110.

Output registry sets

When a registry set is given as instruction output, the value will be placed into all registers specified by the registry set.

Register Bank Bus

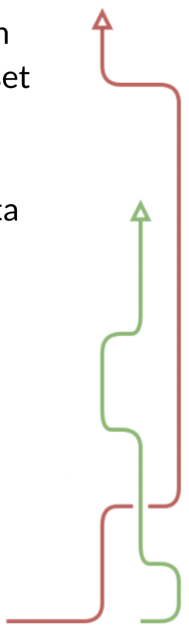
The **microarch** architecture is designed in such a way that simultaneous reading from two different registry sets will often be required. Here is a recommended design for the implementation of CPU busses, that will allow that to be completed within a single clock cycle:



As you can see, there are two register output buses and a single register input bus.

All joints in the register output buses require OR gates on all lines, to ensure proper functioning of output registry set operations.

This ensures that various operations (such as loading data into the ALU or into internal registers used for memory operations) can be completed in a single clock cycle.



Instructions

Instructions in **microarch** have a 3-byte long instruction word. All instructions are also conditional (based on 2 flags set by the CMP instruction).

Encoding

Below is a general instruction encoding scheme for microarch instruction. Exceptions may apply, in which case encoding for a specific instruction will be specified in the instruction listing below.

Op-code	Co1	Co2	Argument-1	Argument-2
4 bit	2-bit	2-bit	8-bit	8-bit

Op-code - Instruction code. Code 1101 is reserved for future use.
Co-1 - Condition 1 (flag ZF)
Co-2 - Condition 2 (flag CF)

Conditions

Each instruction has 2, 2-bit long condition fields (one for the ZF flag and one for the CF flag - those flags are described in further detail in the CMP instruction listing). Condition field works as follows:

True-only bit (older bit)	False-only bit (younger bit)
If 1 - allow true flag value If 0 - forbid true flag value	If 1 - allow false flag value If 0 - forbid false flag value

Therefore:
If the condition is 00 - the instruction will never be executed.
If the condition is 01 - the instruction will be executed only if the flag is set to false.
If the condition is 10 - the instruction will be executed only if the flag is set to true.
If the condition is 11 - the instruction will be executed regardless of the state of the flag.

Instruction listing

Below are described in detail all standard **microarch** instructions.

Naming Convention
Every instruction shall be named using 3 letter mnemonics.

NOP (No operation)

Opcode	Argument 1	Argument 2
0000	Reserved, must be 0	Must be 0
Do nothing. Both arguments must be 0 for the encoding to be valid, therefore the op-code may be used for other instructions in ISA extensions.		

CMP (Compare registers)

Opcode	Argument 1	Argument 2
1111	Input/Output registry set	Input registry set
Subtract input Argument 2 from input Argument 1, write the result to output Argument 1, set the flag CF to the value of the carry bit from the operation, set the flag ZF to 0 if the subtraction result is not equal to 0, set the flag ZF to 1 if the subtraction result is equal to zero.		

ADD (Add registers)

Opcode	Argument 1	Argument 2
1110	Input/Output registry set	Input registry set
Add input Argument 2 to input Argument 1, write the result to output Argument 1		

SET (Set registers to an immediate)

Opcode	Argument 1	Argument 2
0001	Output registry set	8 bit immediate value
Set output registry set (Argument 1) to the immediate value specified by the value in Argument 2.		

MOV (Move between registers)

Opcode	Argument 1	Argument 2
1100	Output registry set	Input registry set
Copy value from input registry set (Argument 2) into the output registry set (Argument 1).		

LDM (Load memory to registers)

Opcode	Argument 1	Argument 2
0010	Output registry set	Input registry set
Load byte from memory from address specified by the input registry set (Argument 2) into the output registry set (Argument 1).		

STM (Store registers in memory)

Opcode	Argument 1	Argument 2
0011	Output registry set	Input registry set
Store byte into memory at the address specified by the output registry set (Argument 1) from the input registry set (Argument 2).		

NAD (Bitwise NAND)

Opcode	Argument 1	Argument 2
1000	Input/Output registry set	Input registry set
Read values from both input registry sets (Argument 1 & 2) bitwise NAND them together and save the result into the output registry set (Argument 1).		

AND (Bitwise AND)

Opcode	Argument 1	Argument 2
1001	Input/Output registry set	Input registry set
Read values from both input registry sets (Argument 1 & 2) bitwise AND them together and save into the result the output registry set (Argument 1).		

XOR (Bitwise XOR)

Opcode	Argument 1	Argument 2
1010	Input/Output registry set	Input registry set
Read values from both input registry sets (Argument 1 & 2) bitwise XOR them together and save into the result the output registry set (Argument 1).		

SHR (Shift N bits right)

Opcode	Argument 1	Argument 2
1011	Input/Output registry set	8 bit, Immediate value
Shift value of an input/output registry set (Argument 1) by an offset given as an immediate value (Argument 2). Only the lower 3 bits of the offset are used, an implementation is allowed to not support offsets larger than 1.		

CID (CPU Identification)

Opcode	Argument 1	Argument 2
0110	Reserved, must be 0	8 bit, Immediate value
Return CPU information and identification. This function can be used to query the CPU for supported extensions and other details. The immediate value (Argument 2) specifies the page number to query. Registers R0 though R3 are modified and R4 through R7 are left as is. For full documentation see the **Identification** section.		

JPR (Jump to address in registers)

Opcode	Argument 1	Argument 2
0100	Input registry set, youngest 8 bits	Input registry set, oldest 8 bits
The two input register values are combined to form a 16 bit value that is stored into the PC register		

JPI (Jump to immediate address)

Opcode	Argument 1	Argument 2
0101	Immediate value, youngest 8 bits	Immediate value, oldest 8 bits
The two immediate values are combined to form a 16 bit value that is stored into the PC register		

CTR (Control)

Opcode	Argument 1	Argument 2
0111	Immediate value. 8 flags.	Immediate value. Reserved for future use.
This instruction stops code execution and depending on the arguments engages a select power saving mode. See the **Power Saving** section for more information on the available flags.		

Memory Map

In the following chapter of this document we specify certain fixed memory addresses, for both data memory and instruction memory, that a **microarch** processor expects will be used for specific applications.

Memory banks

For supporting data memory sizes larger than 256 bytes, memory banks can be utilized.

Current memory bank number should be placed at memory address 0x03, whenever applicable, and synchronized across all banks.

Interrupts

The **microarch** processor core supports a single interrupt line. When an interrupt is received, the processor will finish execution of the current instruction. Then it will store the return address, in Big Endian at addresses 0x00 and 0x01 in the data memory and execute a jump to address 0x0000 in the instruction memory, where the interrupt handler should be implemented.

If the processor allows more than 1 interrupt source, an 8-bit interrupt number should be stored under address 0x02 in data memory. Otherwise the value of that byte after an interrupt is undefined.

If the implementation supports memory banks, those fields should be placed in memory bank 0.

I/O Devices and Interfaces

Control registers for I/O interfaces (both parallel and serial ports) and other devices (for example a system for writing into the microcontroller's flash memory) should be mounted at memory addresses from 0x04 to 0x1F.

If the implementation supports memory banks, those fields should be placed in memory bank 0.

CPU Identification

For each valid CID page number (shown in the first column) the CID instruction reads up to 32 bits of CPU specific data. The table below shows the meaning of particular bits. Extension bits tell if a specific extension is supported. Currency there are no registered extensions.

Page Number	R0	R1	R2	R3
0	Reserved for future use	Extension bits, block A	Extension bits, block B	Extension bits, block C

Power saving

The CTR instruction stops code execution and engages a select power saving mode. Execution is resumed with an interrupt.

Selection is done using 8 flags. Other 8 bits are reserved for use by extensions. All flags are active low - 0xFF will not engage a power saving mode, 0x00 will shut off the device.

Bit	Flag description
0	Powers off the logic (ALU, CU and MMU) and all internal registers, except the Program Counter and the Instruction Registry.
1	Cuts power to the Instruction Registry and pipeline logic (if exists)
2	Cuts power to the programmer’s registers (R0-R7) and erases them.
3	Powers off and invalidates data cache (if exists).
4	Powers off and invalidates instruction cache (if exists).
5	Cuts power to the Data Memory (in microcontrollers) and erases it.
6	Sets low logical state on a select pin (implementation dependent) - may be used to cut power to other devices.
7	Cuts power to the interrupt handling logic and the Program Counter.

Please note, that if flag 7 is engaged, execution will not resume with an interrupt and the device will be shut off until full hardware reset.

Microarch ISA © 2025 by Mateusz Nurczyński, and Maksymilian Goryszewski is licensed under CC BY-SA 4.0.
To view a copy of this license, visit https://creativecommons.org/licenses/by-sa/4.0/