Computer Science

Created: 11.11.2023
Last updated: 13.11.2023
Made by @Pinkulani
https://pinkulani.com

Reduced Instruction Set (RISC)

Reduced Instruction Set Computer (RISC) is a simpler instruction set which means it might require more instructions to achieve the same task compared to the Complex Instruction Set Computer (CISC). RISC instructions are being executed faster compared to CISC because they are simpler code.

Assembly

Common Instructions

These are only 8051 Controller Instructions that I use to get my tasks done there are a lot more. All Registers like A, Rn, DPTR and Addresses can be called upon with any Code but I will only make some examples and not all. Indirect Addresses need to be changed/worked on over the DPTR.

Arithmetic Operations

Operand Code	Operands	Description
INC	A	Increment Accumulator by 1
INC	Rn	Increment Register by 1
DEC	A	Decrement Accumulator by 1
DEC	Rn	Decrement Register by 1
CLR	A	Remove data from Accumulator
SWAP	A	Switch Nibbles of Accumulator
DIV	A	Divide A by B-Register
RL	A	Rotate Data in A left by 1 place
RR	A	Rotate Data in A right by 1 place
SETB	Bit	Set the addressed Bit to 1
CLR	Bit	Set the addressed Bit to 0
ANL	A, Rn	Bitwise AND-Conjunction between A and Rn
ORL	A, Rn	Bitwise OR-Conjunction between A and Rn

Jump Commands

Operand Code	Operands	Description
LJMP	(Adress)	Jump to Adress in 64K-Block
JMP	@A+DPTR	Jump to Adress made out of A and DPTR

JB	Bit, Rel	Jump if Bit is set
JNB	Bit, Rel	Jump if Bit is not set
JNZ	Rel	Jump if data is not 0
DJNZ	Rn, Rel	Decrement data by 1 and jump if not 0
CJNE	Rn, #Data, Rel	Compare Register with Constant and jump if not equal
LCALL	(Adress)	Call subprogram in 64K-Block
RET		End of subprogram
RETI		End of subprogram and delete Interrupt-Flag

Data Transport

Operand Code	**Operands**	**Description**
MOV	Rn, #Data	Load Register with constant
MOV	A, #Data	Load Accumulator with constant
MOVX	A, @DPTR	Copy data from external memory in to Accumulator
MOVX	@DPTR, A	Copy data from Accumulator in to external Memory
MOVC	A, @A+DPTR	Get constant from table in EEPROM
NOP		No activity

Interrupts

In special cases like manufacturing in a factory there needs to be a way to stop the current process really quickly when the emergency switch is pressed. The problem is that these emergency stops are not often used and need to be executed without any delay. There are 2 methods of processing such emergency signals.

Polling

With Polling there is always a loop where it is checked if there has been a Interrupt Signal. First the Program will execute and after finishing it's run it will check if there has been any Signal and then stop or continue and repeat the process and check fi the Interrupt has been activated. A big disadvantage of this is that the program has to finish before it can process the interrupt which might give it a long reaction time.

Interrupt

The interrupt is made by a Hardware Pulse which means that as soon as the Interrupt Signal is made it will be executed after finishing the last machine cycle to not create errors in the program and then go to the Interrupt-Service-Routine (ISR). After executing the last machine cycle the address of the next Command will be saved in the Stack Memory. If the ISR has finished the program will be continued from the saved address.