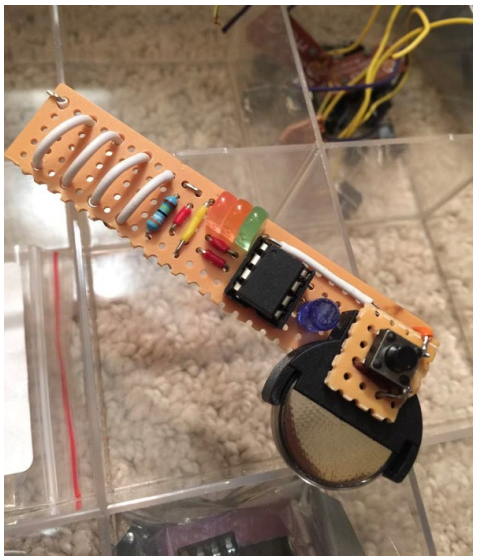
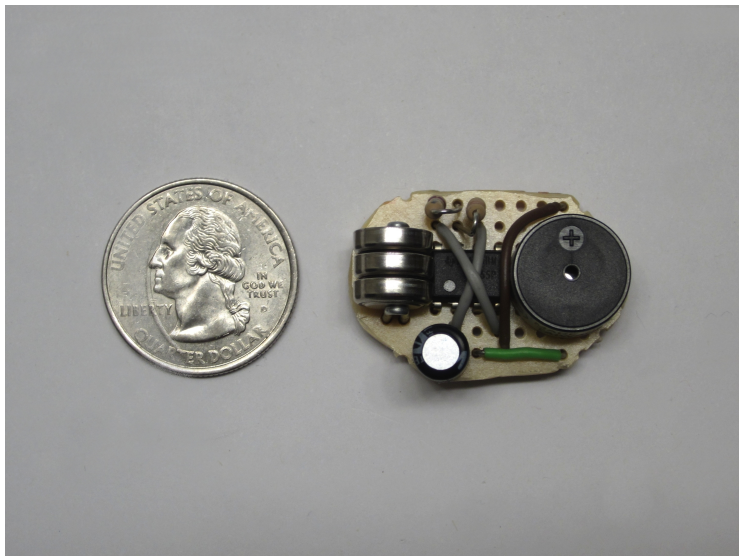
ATtiny Custom CPU

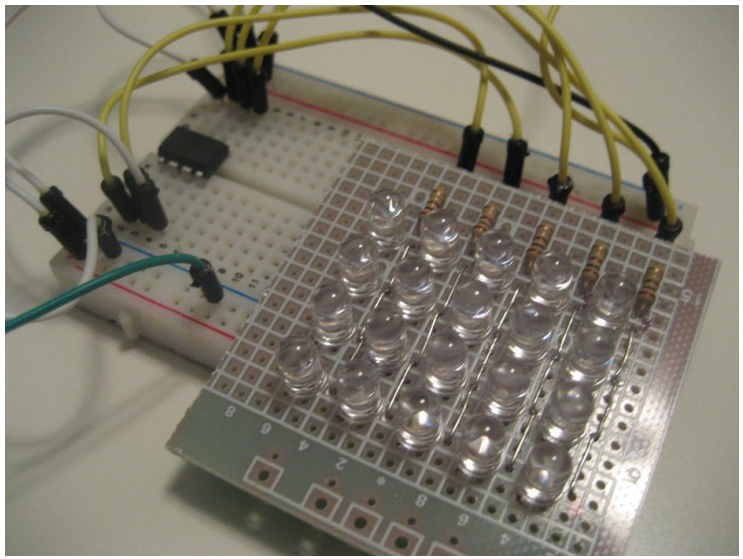
Greg Birge

What is an ATtiny?





What is an ATtiny?



ATTINY POCKET SEQUENCER

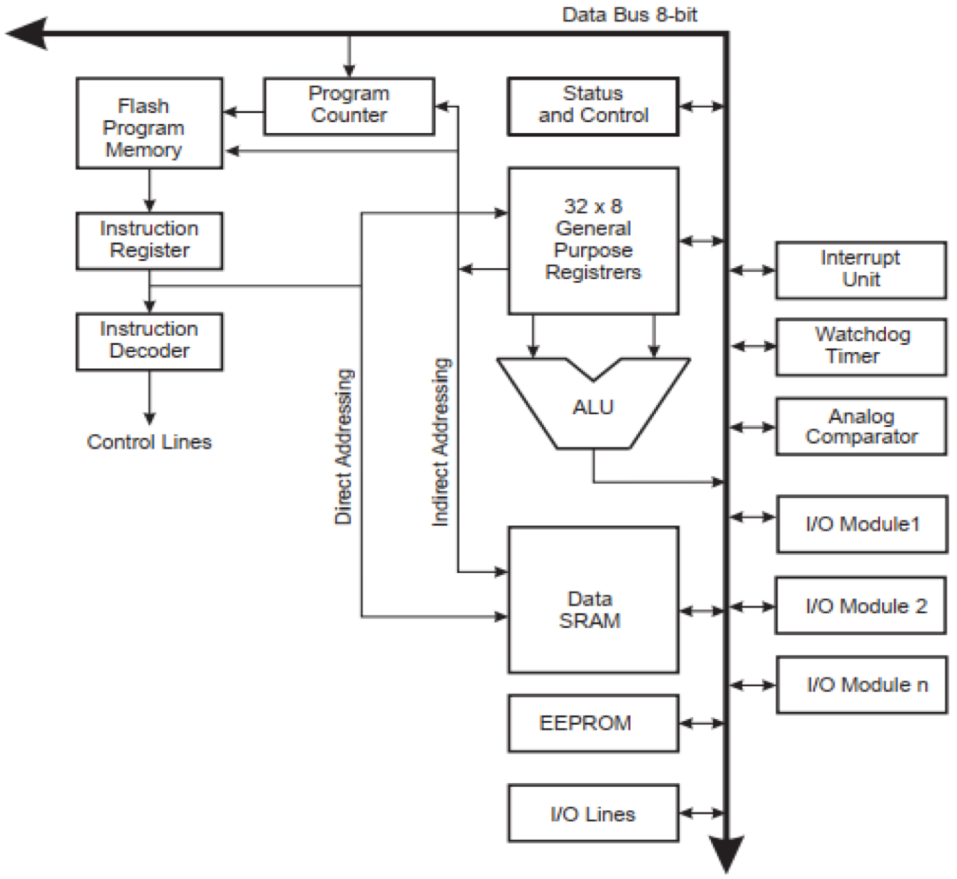
https://www.youtube.com/watch?v=M4ogB8NUQPQ&feature=emb_logo

Project Scope

- Focusing on primary operation with registers and ALU
- Implement Write back and communication with basic I/O over Data-bus
- Ignore VHDL Clock division by using External Clock input

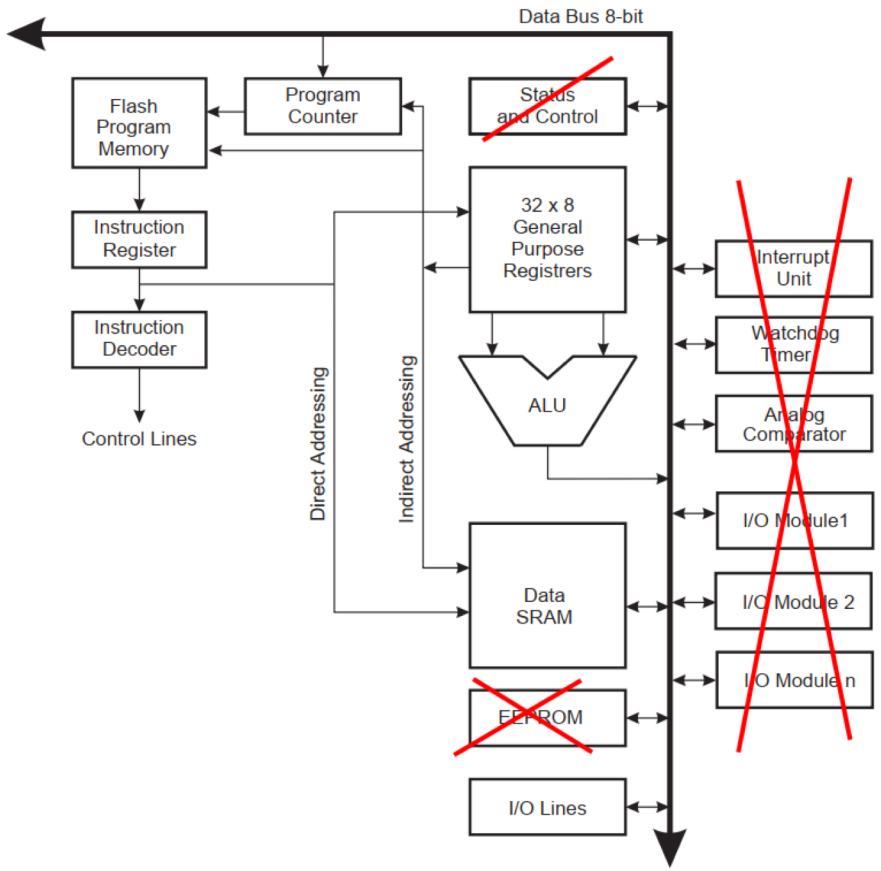
Project Scope

Figure 4-1. Block Diagram of the AVR Architecture



Project Scope

Figure 4-1. Block Diagram of the AVR Architecture



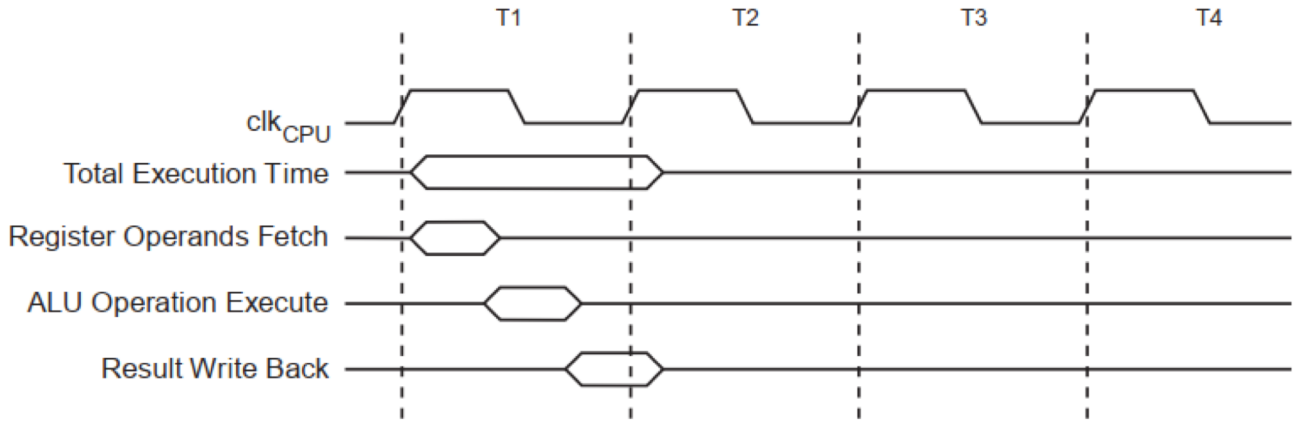
Instruction Set

Instruction Set	Operation	Description	5-bit OP-Code	Field1	Field2	Field3
NOP	No Operation	nothing	"00000"	Null	Null	Null
OR	RDn ← RSn | RDn	OR source & destination registers	"00001"	"00000000000000000"	RSn(5)	RDn(5)
ORI	RDn ← RSn | Data(8)	OR source and 8-bit value	"00010"	"000000000" + Data(8)	Null	RDn(5)
AND	RDn ← RSn * RDn	AND source & destination registers	"00011"	"00000000000000000"	RSn(5)	RDn(5)
ANDI	RDn ← RDn && Data8	And register with Immediate value	"00100"	"000000000" + Data(8)	"00000"	RDn(5)
SUB	RDn ← RSn - RDn	Subtract Source with Dest., store in dest.	"00101"	"00000000000000000"	RSn(5)	RDn(5)
SUBC	RDn ← RSn - RDn -~C_i	Subtract Source with Dest. & carry bit, store in dest.	"00110"	"00000000000000000"	RSn(5)	RDn(5)
ADD	RDn ← RSn + RDn	Add registers together, storing result in destination	"00111"	"00000000000000000"	RSn(5)	RDn(5)
ADDC	RDn ← RSn + RDn + C_i	Add registers together with carry bit, storing result in destination register	"01000"	"00000000000000000"	RSn(5)	RDn(5)
SRA	RDn ← RDn >> Shift left amt.	Shift register right, maintaining signed bit (if MSB is 1, fill left side with 1's after shifting)	"01001"	"000000000" + Shift Amt.	"00000"	RDn(5)
ASR	RDn ← RDn << Shift right amt.	Shifts a register left, zeroes are shifted in	"01010"	"000000000" + Shift Amt.	"00000"	RDn(5)
LDI	Data8 → Rn	Load register (Rn) with given data	"01011"	"000000000" + Data(8)	"00000"	Rn(5)
MOV	RSn → RDn	Copy 8bit data between registers	"01100"	"00000000000000000"	RSn(5)	RDn(5)
XOR	RDn ← RSn xor RDn	xors two registers	"01101"	"00000000000000000"	RSn(5)	RDn(5)
BREQ *	RO ← RSn == RDn	branch if two registers are equal	"01110"	"00000000000000" + RO(5)	RSn(5)	RDn(5)
BRNE *	RO ← RSn != RDn	Branch if Not equal	"01111"	"00000000000000" + RO(5)	RSn(5)	RDn(5)
ADIW *	RDn ← RSn + Data16	ADD 16 bit word to two registers	"10000"	Data(16)		RDn(5)
SBIW *	RDn ← RSn - Data16	SUB 16 bit word from two registers	"10001"	Data(16)		RDn(5)

Instruction Timing

Figure 4-5 shows the internal timing concept for the Register File. In a single clock cycle an ALU operation u two register operands is executed, and the result is stored back to the destination register.

Figure 4-5. Single Cycle ALU Operation



Pipelining

Figure 4-4. The Parallel Instruction Fetches and Instruction Executions

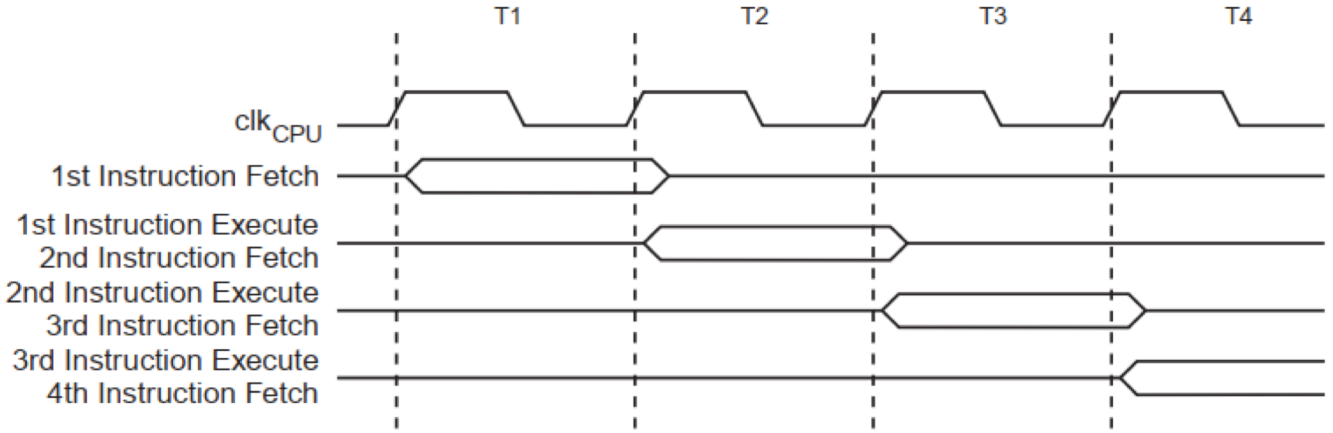


Figure 4-4 shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast access Register File concept. This is the basic pipelining concept to obtain up to 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

Next Steps

- Finish Coding
- Testing
- Implement Rudimentary I/O
- Pipelining or two cycle instructions
  - Decision for future Greg

Sources

Sources pt. 2

- https://makezine.com/projects/miniature-beeping-circuit-prank/
- https://www.instructables.com/id/Attiny-Pocket-Sequencer/
- https://www.instructables.com/id/Creating-a-charlieplexed-LED-grid-to-run-on-ATTiny/
- https://www.instructables.com/id/Attiny-Canbot/
- https://www.instructables.com/id/ATTiny-EMF-Detector/
- https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet.pdf

Questions

Please no.