SuperPIK08 - ein elementarer Mikrocontroller

Vorläufige Kurzübersicht

Stand: 0.31 vom 25. 6. 04

Diese Schrift ist ein verkürzter und abgewandelter Auszug aus der Vorläufigen Kurzbeschreibung. Sie betrifft die vom Emulator Stand 0.x unterstützte Basismaschine mit Programmspeicherlesezusatz.

1. Einführung

SuperPIK08 ist ein einfacher Mikrocontroller, der einer Industriestandard-Architektur nachgebildet ist, nämlich den PIC16-Typen der Fa. Microchip.

Einsatzzwecke:

- als Lehrmittel zur Einführung in die Mikrocntrollerprogrammierung,
- zu vergleichenden Studien (Bewertung verschiedenartiger Architekturen),
- in sog. IP-Cores zum Einbau in Systemlösungen auf Grundlage programmierbarer Logikschaltkreise (FPGAs).

Erste Implementierung:
Emulation auf PC mit echten E-A-Schnittstellen. Geschwindigkeit entspricht der eines Mikrocontrollers mit einigen MHz Taktfrequenz.

Programmierphilosophie und Programmiermodell
Es geht um kostengünstige Hardware für Steuerungsaufgaben, die nicht allzu rechenintensiv sind, mit anderen Worten: um Embedded Systems im unteren Preis- und Leistungsbereich.

Typische (vorteilhafte) Merkmale der PIC16-Architektur (Microchip):

- einfacher, überschaubarer Befehlsvorrat,
- Variable und E-A-Schnittstellen befinden sich in einem gemeinsamen Adreßraum,
- die Befehle arbeiten direkt mit den Variablen bzw. E.-A-Belegungen. Damit enfällt die Notwendigkeit, diese Parameter aus einem RAM und aus der E-A-Hardware eigens in einen Registersatz zu laden (LOAD/STORE und IN/OUT-Befehle sind nicht erforderlich).

Abweichungen von der PIC16-Architektur:

- 512 Speicherstellen (Register) statt 256,
- Befehlslänge: 16 Bits statt 12 oder 14,
- zusätzliche Maschinenbefehle,
- verbesserte indirekte Registeradressierung,
- Befehlsspeicher bereits in Grundausstattung maximal 8k Worte. Keine Einschränkungen bei Verzweigung und Unterprogrammruf.
- Verbesserungen beim Datenlesen aus dem Programmspeicher,
- grundsätzlich ist die Programmierung weniger umständlich und auch nicht von kleinlichen Einschränkungen betroffen.

Baukastensystem
SuperPIK08 ist praktisch ein Baukasten aus verschiedenen Moduln, die je nach Anforderung miteinander kombiniert werden können (das betrifft sowohl die Generierung anwendungsspezifischer Hardware als auch die Freischaltung verschiedener Funktionsmerkmale im Emulator):

- Basismaschine (Basic Processor Core),
- Indirektzugriffszusatz (Indirect Access Feature),
- Relocationszusatz (Relocation Feature),
- Programmspeicherzugriffszusatz (Program Memory Access Feature),
- Programmspeicherschreibzusatz (Program Memory Write Feature),
- erweiterte Verarbeitungsfunktionen (Extented Processor Operations),
- erweiterte Stackfunktionen (Extented Stack Operations),
- Break-In-Zusatz (Break In Feature),
- Hardware-Direktsteuerzusatz (Direct Control Feature),
- Mehrmaschinenzusatz (MVM Feature),
- Stromsparzusatz (Power Savings Feature),
- Fehlersuchzusatz (Debugging Feature).

2. Die Basismaschine

2.1. Grundsätzliche Merkmale

Architekturprinzip: Einadreßmaschine.

Speicherorganisation: Harvard.

Verarbeitungsbreite: 8 Bits.

Befehlslänge: 16 Bits.

Adressierungsvermögen der Befehlsadressierung: 8k 16-Bit-Worte.

Adressierungsvermögen der Daten- und E-A-Adressierung: 512 Bytes.

Programmspeicher: ROM.

Datenspeicher: Universalregistersatz. In diesem Registersatz sind zusammengefaßt:

- universell verwendbare Register (RAM-Zellen),
- programmseitig zugängliche Hardwareregister des Prozessors,
- programmseitig zugängliche Register der E-A-Hardware.

Erreichbarkeit der E-A-Hardware: über den Adreßaum des Datenspeichers.

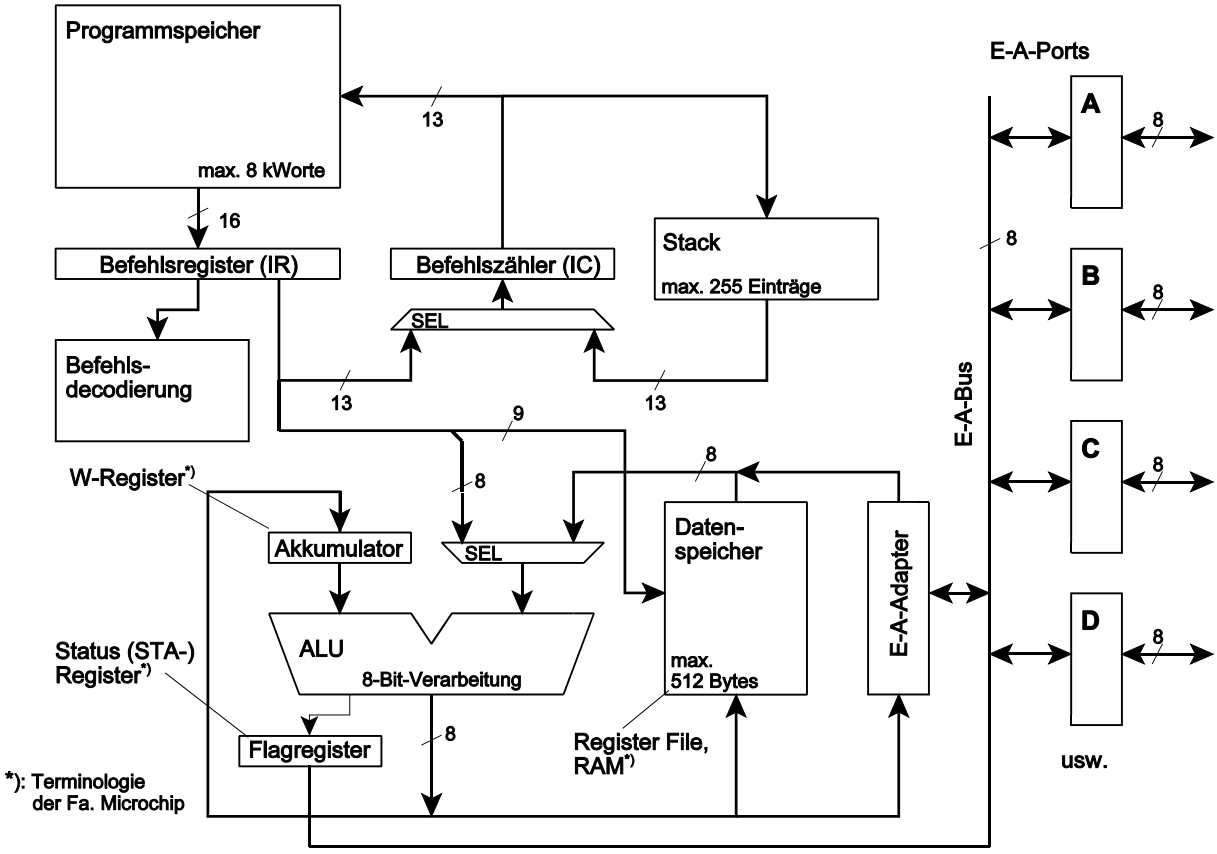
Adreßrettung bei Unterprogrammruf: über Hardware-Stack.

Stacktiefe: maschinenspezifisch (z. B. 16 oder 31). Obergrenze = 255.

Was es NICHT gibt:

- indirekte Registeradressierung,
- berechnete Verzweigungsadressen,
- Interrupts.

2.2. Hardwarestruktur



2.3. Programmiermodell

Verarbeitungsoperationen
Die Arithmetik-Logik-Einheit ALU verknüpft den Inhalt des Akkumulators (W-Register, WREG) mit dem Inhalt der adressierten Datenspeicherzelle (Register REG) und schreibt das Ergebnis wahlweise in den Akkumulator oder in die Datenspeicherzelle zurück. Je nach Operation werden verschiedene Bits im Flagregister (STA-Register) gestellt.

Operationsschema:

<WREG> OP <REG> => <WREG> oder <WREG> OP <REG> => <REG>

Direktwertoperationen
Die betreffenden Befehle enthalten einen 8-Bit-Direktwert (LITERAL). Dieser Direktwert wird mit dem Inhalt des Akkumulators (W-Register) verknüpft. Das Ergebnis wird in den Akkumulator zurückgeschrieben. Je nach Operation werden verschiedene Bits im Flagregister (STA-Register) gestellt.

Operationsschema:

<WREG> OP <LITERAL> => <WREG>

Laden und Speichern
Der Akkumulator (W-Register) kann wahlweise mit einem Direktwert oder mit dem Inhalt einer adressierten Datenspeicherzelle (Register REG) geladen werden. Des weiteren ist es möglich, den Akkumulatorinhalt in eine adressierte Datenspeicherzelle zu transportieren.

Operationsschemata:

LITERAL => <WREG>; <WREG> => <REG>; <REG> => <WREG>

Einzelbitoperationen
Jede einzelne Bitposition des Datenspeichers läßt sich abfragen und auf Null oder auf Eins setzen. Bitadressierung: durch Direktwerte in den entsprechenden Befehlen.

Unbedingte Verzweigungen
Mittels Befehl GOTO kann jede beliebige Adresse des Programmspeichers angesprungen werden. Adressierungsvermögen: 8k Worte.

Bedingte Verzweigungen
Es gibt Bitafragebefehle mit bedingtem Überspringen (SKIP) des Folgebefehls. Durch Kombination mit unbedingten Verzweigungen lassen sich beliebige bedingte Verzweigungen programmieren. Hierbei kann jedes Bit im Datenspeicher als Verzweigungsbedingung dienen.

Abfrage der im Flagregister (STA) enthaltenen Bedingungsbits
Das Flagregister ist über eine feste Datenspeicheradresse zugänglich und kann mit Bitabfragebefehlen ausgewertet werden.

Unterprogrammruf
Mittels Befehl CALL kann jede beliebige Adresse des Programmspeichers angesprungen werden. Adressierungsvermögen: 8k Worte.

Rettung der Rückkehradesse
In einem Hardwarestack, der programmseitig nicht beeinflußbar ist. Es ist nur eine echtes Schachteln von Unterprogrammrufen zulässig (Rückkehrreihenfolge = umgekehrte Aufrufreihenfolge).

Datenspeicheradressierung
Durch Direktwertangaben im Befehl. Adressierungsvermögen: 512 Bytes

2.4. Programmspeicherlesezusatz

Wirkunge:
Lesen von Daten (Konstanten) aus dem Programmspeicher mit berechneten Adressen:

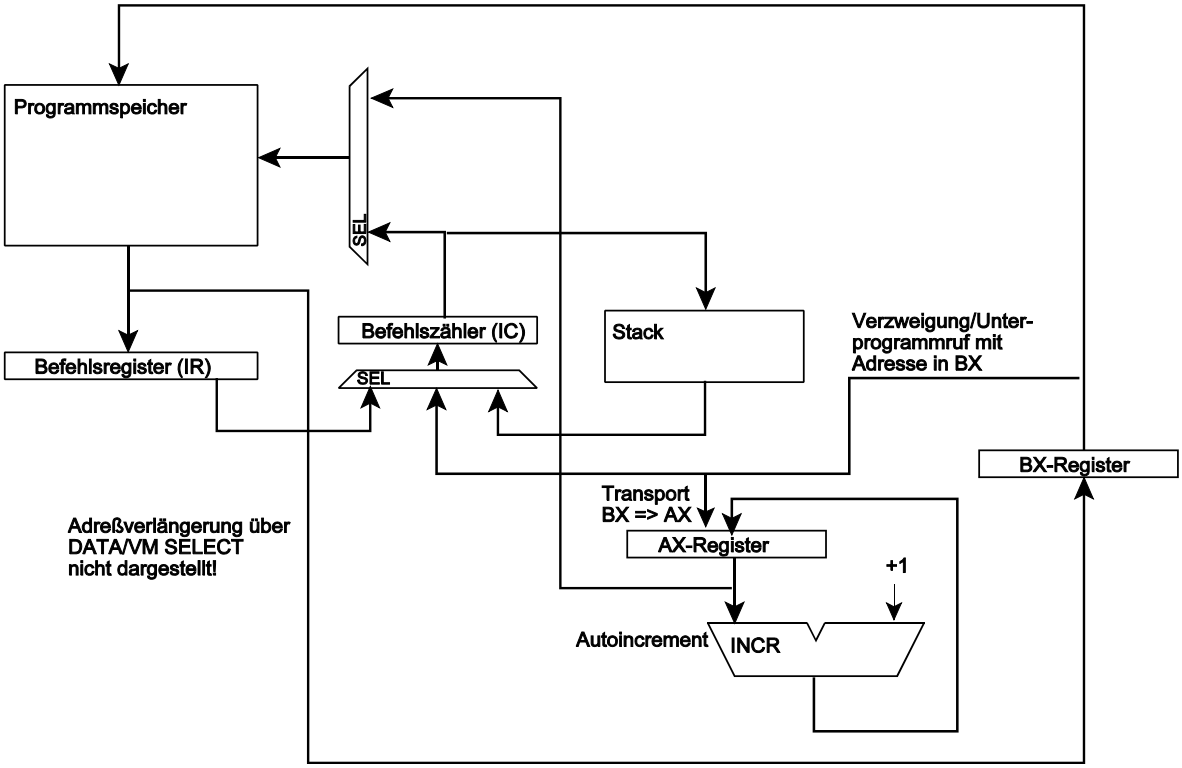
Technische Lösung:
Ein zusätzliches 16-Bit-Register AX (Programmspeicheradreßregister).

Programmseitige Zugänglichkeit: über die Datenspeicheradressen 4 und 5.

Zugriffsschema:

Befehl MOVPFL: <<AX, Bits 7...0>> => Register F

Befehl MOVPFH: <<AX, Bits 15...8>> => Register F. Anschließend wird der Inhalt von AX um Eins erhöht (Adreßzählung).



2.5. Datenspeicherbelegung

Adreßbereich: von 0 bis 511 (0H...1FFH).

Adreßbereich	Belegung
0H...09H (10 Bytes)	Hardware. Nicht verwenden. In der erweiterten Basismaschine sind nur die Adressen 4, 5 und 8 zugänglich (Register AX und STA)
0AH...3FH (54 Bytes)	erster Arbeitsbereich. Frei verfügbar
40H...1FFH	Arbeits- und E-A-Bereiche. Belegung modellspezifisch

Datenspeicherbelegung 0H...09H:

		7	6	5	4	3	2	1	0
0	res.								
1	res.								
2	res.								
3	res.								
4	PGM ADRS Low (AXL)								
5	PGM ADRS High (AXH)								
6	res.								
7	res.								
8	Status	S	Z	res.	D	O	res.	res.	C
9	res.								

E-A-Adressen
Die Belegung ist modellspezifisch. Im Emulator Stand 0.x ist der Adreßbereich von 40H bis 7FH für E-A-Zwecke reserviert (64 Adressen). Die Belegung hängt vom angeschlossenen Portadapter ab.

Belegungsbeispiel (15 universelle E-A-Ports zu 8 Bits):

E-A-Port	**Adresse**
Port A, Richtung	40H
Port A, Daten	41H
Port A, Anschlüsse (nur lesen)	42H
Port B, Richtung	43H
Port B, Daten	44H
Port B, Anschlüsse (nur lesen)	45H
Port C, Richtung	46H
Port C, Daten	47H
Port C, Anschlüsse (nur lesen)	48H
Port D, Richtung	49H
Port D, Daten	4AH
Port D, Anschlüsse (nur lesen)	4BH
Port E, Richtung	4CH
Port E, Daten	4DH
Port E, Anschlüsse (nur lesen)	4EH
Port F, Richtung	4FH
Port F, Daten	50H

E-A-Port	Adresse
Port F, Anschlüsse (nur lesen)	51H
Port G, Richtung	52H
Port G, Daten	53H
Port G, Anschlüsse (nur lesen)	54H
Port H, Richtung	55H
Port H, Daten	56H
Port H, Anschlüsse (nur lesen)	57H
Port I, Richtung	58H
Port I, Daten	59H
Port I, Anschlüsse (nur lesen)	5AH
Port J, Richtung	5BH
Port J, Daten	5CH
Port J, Anschlüsse (nur lesen)	5DH
Port K, Richtung	5EH
Port K, Daten	5FH
Port K, Anschlüsse (nur lesen)	60H
Port L, Richtung (PORT A)*)	61H
Port L, Daten (PORT A)*)	62H
Port L, Anschlüsse (nur lesen) (PORT A)*)	63H
Port M, Richtung (PORT B)*)	64H
Port M, Daten (PORT B)*)	65H
Port M, Anschlüsse (nur lesen) (PORT B)*)	66H
Port N, Richtung (PORT C)*)	67H
Port N, Daten (PORT C)*)	68H
Port N, Anschlüsse (nur lesen) (PORT C)*)	69H
Port O, Richtung (PORT D)*)	6AH
Port O, Daten (PORT D)*)	6BH
Port O, Anschlüsse (nur lesen) (PORT D)*)	6CH

*): Bezeichnungen gemäß Puffer/Portadapter-Kombination PPKI 04a

Das Flagregister (STA-Register):

	7	6	5	4	3	2	1	0
Status	**S**	**Z**	**res.**	**D**	**O**	**res.**	**res.**	**C**

Bit	Bezeichnung		Bedeutung	
7	S	Sign	Vorzeichen.	0: positiv, 1: negativ
6	Z	Zero	Null.	0:Ergebnis ... Null, 1: Ergebnis = Null
4	D	Decimal Carry	Übertrag von Bitposition 3 in Bitposition 4 (BCD-Rechnung)	
3	O	Overflow	Überlauf	
0	C	Carry	Ausgangsübertrag	

Zu den Übertragsbits (C, D)
Sie geben stets an daß ggf. in der jeweils nächst-höherwertigen Stelle weiterzurechnen ist. Beim Addieren handelt es sich um den Übertrag im eigentliche Sinne (Carry), beim Subtrahieren um ein Borgen (Borrow).

Zweierkomplementrechnung und Übertragsbits
Die ALU rechnet im Zweierkomplement. Der hardwareseitige Ausgangsübertrag (CARRY_OUT) wird folgendermaßen zum Bilden des entsprechenden Übertrags-Flagbits (C, D) verwendet:

- Addieren: CARRY OUT => Übertrags-Flag
- Subtrahieren: negiertes CARRY_OUT => Übertrags-Flag.

Beim Rechnen mit vorzeichenlosen Zahlen zeigt ein gesetztes Übertrags-Flag stets (unabhängig von der Rechenoperation) ein Verlassen (Über- oder Unterschreiten) des Wertebereichs an.

2.6. Befehlsübersicht

Mnemonic	Parameter	Beschreibung	Flagbits
Registerbefehle			
ADDWF	F, D	Addieren WREG und Registerinhalt; Speichern in WREG oder Register	S, Z, D, O, C
ANDWF	F, D	bitweise UND-Verknüpfung von WREG und Registerinhalt; Speichern in WREG oder Register	Z
CLRF	F	Register löschen	Z => 1
CLRW	-	WREG löschen	Z => 1
COMF	F, D	bitweise Negation des Registerinhalts; Speichern in WREG oder Register	Z
DECF	F, D	Registerinhalt um Eins vermindern; Speichern in WREG oder Register	Z
DECFSZ	F, D	Registerinhalt um Eins vermindern; Speichern in WREG oder Register. Ist das Ergebnis = 0, den folgenden Befehl überspringen	Z
INCF	F, D	Registerinhalt um Eins erhöhen; Speichern in WREG oder Register	Z
INCFSZ	F, D	Registerinhalt um Eins erhöhen; Speichern in WREG oder Register. Ist das Ergebnis = 0, den folgenden Befehl überspringen	Z
IORWF	F, D	bitweise ODER-Verknüpfung von WREG und Registerinhalt; Speichern in WREG oder Register	Z
MOVF	F, D	Registerinhalt holen und in WREG oder Register speichern, dabei überprüfen, ob der Inhalt = 0 ist	Z
MOVWF	F	WREG im Register speichern	-
RLF	F, D	Registerinhalt um ein Bit über das Carry-Flagbit nach links rotieren; Speichern in WREG oder Register	C
RRF	F, D	Registerinhalt um ein Bit über das Carry-Flagbit nach rechts rotieren; Speichern in WREG oder Register	C
SUBWF	F, D	WREG vom Registerinhalt subtrahieren (F - W); Speichern in WREG oder Register	S, Z, D, O, C
SWAPF	F, D	die beiden Tetraden des Registerinhalts untereinander tauschen; Speichern in WREG oder Register	-
XORWF	F, D	bitweise XOR-Verknüpfung von WREG und Registerinhalt; Speichern in WREG oder Register	Z
Bitbefehle			
BCF	B, F	die adressierte Bitposition im Datenspeicher löschen	-
BSF	B, F	die adressierte Bitposition im Datenspeicher setzen	-

Mnemonic	Parameter	Beschreibung	Flagbits
BTFSC	B, F	die adressierte Bitposition im Datenspeicher abfragen. Ist das Bit = 0, den folgenden Befehl überspringen	-
BTFSS	B, F	die adressierte Bitposition im Datenspeicher abfragen. Ist das Bit = 1, den folgenden Befehl überspringen	-
Direktwertbefehle			
ADDLW	L	Addieren WREG und Direktwert; Speichern in WREG	S, Z, D, O, C
ANDLW	L	bitweise UND-Verknüpfung von WREG und Direktwert; Speichern in WREG	Z
IORLW	L	bitweise ODER-Verknüpfung von WREG und Direktwert; Speichern in WREG	Z
MOVLW	L	Direktwert ins WREG laden	-
SUBLW	L	Subtrahieren WREG vom Direktwert (L - W); Speichern in WREG	S, Z, D, O, C
XORLW	L	bitweise XOR-Verknüpfung von WREG und Direktwert; Speichern in WREG	Z
Programmspeicherlesebefehle			
MOVPFL	F	Lesen aus Programmspeicher. Das niedere Byte des vom Register AX adressierten Speicherwortes wird in das Register F transportiert	-
MOVPFH	F	Lesen aus Programmspeicher. Das höhere Byte des vom Register AX adressierten Speicherwortes wird in das Register F transportiert. Anschließend wird der Inhalt von AX um 1 erhöht	-
sonstige Befehle			
CALL	ADRS	Unterprogrammruf	-
GOTO	ADRS	unbedingte Verzweigung	-
NOP	-	keine Wirkung	-
RET	-	Rückkehr aus Unterprogramm	-

Parameter:

- B = Bitadresse im Byte (3 Bits),
- F = Registeradresse (9 Bits),
- D = Bestimmung des Resultats (Destination; 1 Bit):
  - 0: ins W-Register,
  - 1: in den Datenspeicher.
- L = : Direktwert (8 Bits).