Interner Bericht
DESY F56-78/01
September 1978

BIPOLARE, SCHNELLE MIKROPROZESSORSYSTEME

von

H.-J. Stuckenberg

BIPOLARE, SCHNELLE MIKROPROZESSORSYSTEME

von

H.-J. Stuckenberg
Deutsches Elektronen-Synchrotron DESY, Hamburg

1. Einführung

Es gibt 2 Arten von Mikroprozessoren:

- Slice Prozessoren, sogenannte Bitscheiben-Prozessoren
- normale, wortorganisierte Mikroprozessoren auf einem Chip

Sie unterscheiden sich in 3 wesentlichen Punkten:

- Architektur (1)
- Instruktionssatz (2)
- Wortlänge (3)

(1) Ein Einchip-Mikroprozessor enthält alle Funktionselemente in einem Gehäuse, der Slice-Prozessor hat die Funktionselemente in verschiedene Funktionsblöcke in getrennten Gehäuse unterteilt.

(2) Der Einchip-Mikroprozessor hat einen festen Instruktionssatz, der Slice-Prozessor hat keinen Instruktionssatz im üblichen Sinn, er ist mikroprogrammiert, seine Instruktionen stehen in einem Kontrollspeicher. Der System Entwickler bestimmt während des Schreibens des Mikroprogramms einen meist anwendungsbezogenen Instruktionssatz, oft auch zusätzlich einen festen, nicht von der Anwendung bestimmten Satz, der vom Benutzer nicht geändert werden kann.

(3) Ein Einchip Mikroprozessor hat eine feste Wortlänge, z.B. 8 oder 16 bit. Beim Slice-Prozessor können mehrere Slices kaskadiert werden, daraus ergibt sich eine fast beliebige Wortlänge.

Durch diese Eigenschaften ist das Slice Prozessorsystem meist flexibler in Architektur und Operation, es benötigt aber mehr IC-Gehäuse, mehr Power, höhere Kosten und längere Entwicklungszeit, dagegen braucht der komplette Einchip Mikroprozessor mehr Ausführungszeit, um dieselbe Aufgabe zu lösen, d.h. Bitscheiben Prozessoren sind schneller.

Es gibt auch Unterschiede in der Software-Entwicklung. Für normale Mikroprozessoren mit einem festen Instruktionssatz stehen heute Cross Assembler für höhere Programmiersprachen zur Verfügung, die auf 16- oder 32-bit Maschinen laufen und die einen Objektcode erzeugen, der ein Binärpattern ist, meist allerdings in Oktal oder Hexadezimalcode ausgedruckt. Dieses ist in normalen Rechnersystemen der niedrigste Programmierlevel, für die Mikroprogrammierung in den Slicesystemen ist er immer noch der Makroprogrammlevel. Um eine Makroinstruktion dieser Art im Sliceprozessor auszuführen, werden eine oder mehrere Mikroinstruktionen benötigt.

Die meisten mikroprogrammierten Systeme sind entweder Kontrollersysteme oder Emulatorsysteme.

- Kontrollersysteme haben keinen Instruktionssatz üblicher Art, das gesamte Programm steht im Mikroprogrammspeicher. Sie kontrollieren direkt die externe Hardware.
- Emulatorsysteme haben einen Instruktionssatz, der im Mikroprogrammspeicher implementiert ist. Das Makroprogramm des Benutzers steht in einem externen Speicher. Sie emulieren oder simulieren den Instruktionssatz eines existierenden Rechners oder ein Subset davon.

Eine typisches verallgemeinertes Slicesystem zeigt Bild 1. Die abgebildeten 6 Funktionsblöcke werden im Folgenden beschrieben:

1. CONTROL MEMORY	es enthält die Mikroinstruktion, die die ALU und andere Systemblöcke kontrolliert.
2. MEMORY CONTROL UNIT	sie bestimmt die Adresse der Mikroinstruktion, (auch Sequenzer genannt).
3. ALU	zur Ausführung der arithmetischen und logischen Funktion mit Daten,
4. REGISTER FILE	als Temporärer Speicher für Daten, die in der ALU behandelt werden sollen oder behandelt sind
5. I/O oder MEMORY INTERFACE	als Interface zwischen dem System und dem externen Speicher oder anderer Hardware
6. TIMING	diese Clockfunktion kontrolliert das Latchen der Daten in die verschiedenen Systemregister

2. Kontrollspeicher

Ein Kontroll- oder Mikroprogrammspeicher ist ein N Worte mal M bit Speicher, der die verschiedenen Mikroinstruktionen enthält. Bild 2 zeigt den Speicher. Die Adressen gehen von 0 bis N-1. Jedes Wort enthält M bits, es ist in verschiedene Felder zerbrochen.

Im Bild sind als Beispiel 32 bit gezeichnet, Feld 1 ist 5 bit breit, Feld 2 ist 8 bit breit usw. Das Bild zeigt also das Format einer 32 bit Mikroinstruktion.

Als Beispiel einer Aufteilung möge gelten:

Feld 1 Allg. Verwendung
Feld 2 Branch Adresse
Feld 3 Kontrollbits der nächsten Adresse
Feld 4 Interrupt Kontrollbits
Feld 5 Fast/Slow Clock Selektierung
Feld 6 Carry Kontrollbits
Feld 7 ALU Source Operandenadresse
Feld 8 ALU Funktion
Feld 9 ALU Destination Adresse

3. Speicherkontrolleinheit

Wenn das Format festgelegt ist, muß beschrieben werden, wie die Mikroinstruktionsfolgen ablaufen. Im einfachsten Fall benötigt man einen Mikroprogramm Adresszähler, wie es Bild 3 zeigt.

Mit jedem Clocktakt wird der Inhalt des Zählers inkrementiert, dadurch die nächste Mikroinstruktion aufgerufen. Diese Art ist nicht sehr flexibel, das System kann nur eine feste Folge von Mikroinstruktionen ausführen. Die Inkrementschritte nennt man

CONTINUE oder EXECUTE.

Wenn die Mikroprogrammeinheit auch andere als die nächste Instruktion aufrufen soll, muß sie in der Lage sein, eine BRANCH (Verzweigung) oder JUMP (Spring)-Adresse zu laden. Bild 4 zeigt diese Architektur, wo eine Verzweigungsadresse parallel in den Mikroprogramm Adresszähler geladen wird. Die Ladekontrolle geht durch das 1 bit Feld (Load Enable bit). Ist das Bit 0, kann keine Verzweigungsadresse geladen werden, ist es 1, ist das Laden frei, und die im Mikroprogramm stehende Verzweigungsadresse wird in den Adresszähler geladen. Dadurch wird eine N-fache Verzweigung möglich.

Durch die BRANCH- und JUMP Instruktion ist die Folge von Mikroinstruktionen flexibler geworden, was fehlt, ist noch die Möglichkeit, über Entscheidungen bedingte Verzweigungen einzuführen. Bild 5 zeigt eine solche Kontrolleranordnung für 2 verschiedene Bedingungen.

Das Ladeselektierfeld ist jetzt 2 bit breit, es kontrolliert einen Multiplexer mit 4 Eingängen auf folgende Weise:

S1	S0	Funktion	
0	0	Continue	
0	1	Jump if Cond. 1 is true	CONDITIONAL BRANCHING
1	0	Jump if Cond. 2 is true	
1	1	Jump if unconditional	

Sind S1, S0 = 0,0, wird die Ladekontrolle inaktiv, d.h. das CONTINUE-Programm wird ausgeführt. Sind S1, S0 = 0,1 wird der D1-Eingang selektiert. Ist dabei Cond. 1 = 0 , wird der Adresszähler inkrementiert, ist Cond. 1 = 1, wird die Branch Adresse im nächsten Clockzyklus parallel eingeladen. Ist S1, S0 = 1,0, wird D2 selektiert, hier gilt das gleiche. Ist S1, S0 = 1,1 wird D3 selektiert, d.h. logic High auf load gelegt, das bedeutet, daß die Branch Adresse im nächsten Zyklus unbedingt eingeladen wird.

Dieses Beispiel zeigte die Wirkung der Statements CONTINUE, CONDITIONAL BRANCH, UNCONDITIONAL BRANCH.

Schauen wir uns die Konfiguration im nächsten Bild 6 an, so sehen wir 2 Unterschiede zum vorangegangenen Bild, nämlich ein Pipelineregister und das Ersetzen des Adresscounters durch eine Kombination von Incrementer und Register. Das Pipelineregister enthält die Mikroinstruktion, die gerade in der Maschine ausgeführt wird. Gleichzeitig aber wird die Adresse der nächsten Mikroinstruktion an den Mikroprogrammspeicher gelegt, damit die nächste Instruktion geholt und an den Eingang des Pipelineregisters gelegt. Durch diese Überlappungstechnik wird die Zugriffszeit zum Mikroprogrammspeicher in einer getrennten Zeitschleife behandelt, unabhängig von anderen Teilen der Maschine.

Man muß aber beachten, daß in den verschiedenen Registern des Systems Informationen aus dem vorangegangenen Zyklus stehen, in anderen die des laufenden Zyklus, in weiteren die des kommenden Zyklus. Die zweite Änderung betrifft die Hinzufügung des Adress-MUX an der Quelle der Mikroprogramm Adresse. Damit kann entweder das Pipelineregister oder das Mikroprogramm-Zählerregister die nächste Adresse bestimmen. Dieser Adress-MUX wird vom Condition Code-MUX gesteuert. Wird aus dem Pipelineregister als nächste Adresse eine Verzweigungsadresse angelegt, kann am Ausgang des Adress-MUX der Incrementer die "Adresse + 1" als nächstfolgende erzeugen und über das Mikroprogrammregister und den Adress-MUX einladen. Das gleiche gilt, wenn das Mikroprogrammregister über den Adress-MUX ausgewählt war, die nächste Adresse zu erzeugen, auch dann wird der Inkrementer als folgende "Adresse A+1" über das Register einlesen, d.h. der Inkrementer ist für beide Datenquellen wirksam.

Auch in Mikroprogrammen werden Subroutinen vorkommen, d.h. eine einzelne oder ein Satz von Mikroinstruktionen sollen von verschiedenen Programmteilen benutzt werden können. Wir benötigen dazu ein Adressregister, in dem die Rücksprungadresse steht, von der aus das Mikroprogramm nach Beendigung der Subroutine weiter geführt werden kann.

Dies ist in Bild 7 gezeigt, in dem zusätzlich zum bisherigen ein "Subroutinen and Loop Stack" eingeführt wird zusammen mit einem Stackpointer. Die Kontrollsignale hierfür werden durch einen Baustein, der als "Next Address Control" bezeichnet wird, bereitgestellt. Der Ausgang des Stacks, d.h. die zuletzt hingeschriebene Rücksprungadresse kann über den Adress-MUX als nächste Adresse angewählt werden.

Der Condition Code MUX steuert ebenfalls über den Next Address Control-Baustein die Quelle der nächsten Programmadresse an. Am Eingang des Condition Code MUX ist ein Programmschleifenzähler, der gesetzt werden kann auf eine Anzahl von Schleifendurchläufen. Jede ausgeführte Schleife zählt den Zähler herunter, sein Nulldurchgang läßt die nächste Programmspeicheradresse springen. Ein solcher Mikroprogrammsequenzer wird von mehreren Firmen vertrieben, die typischen Durchlaufzeiten durch den Sequenzer sind etwa 50 ns.

Um schnellere Systeme aufzubauen, muß man ECL-Bausteine verwenden. MOTOROLA hat in der 10800-Serie ebenfalls einen Mikroprogrammsequenzer aufgebaut, 10801 dessen komplexere und flexiblere Struktur wir jetzt ansehen wollen.

Bild 8 oben zeigt das Blockdiagramm des ECL-Sequenzer 10801 und wir erkennen die starke Ähnlichkeit zur vorigen TTL-Version. Es gibt jedoch wesentlich mehr Möglichkeiten, den Baustein einzusetzen. Er enthält ein 4 bit breites Mikroprogrammspeicher-Adressregister, CRO genannt, das im mittleren Bild 8 genauer

zu sehen ist. In diesem Bild sind auch die Busanschlüsse eingezeichnet, die im 10800-System üblichen I- und Ø-Busse, die bidirektional aufgebaut sind.

Der Next Adress Generator, der durch verschiedene Signale kontrolliert wird, selektiert die Adresse der nächsten Mikroinstruktion und schaltet sie in das CRO-Register.

Der 10801 hat 16 Instruktionen, die den Programmfluß kontrollieren. Diese Befehle schließen das Inkrementieren der laufenden Adresse sowie verschiedene Sprungoperationen ein. Die Quellen, aus denen die Sprungadressen kommen können, sind der I-Bus, der Ø-Bus, der Next Adress Bus (NA) sowie Register CR1 und CR2. (s. Bild 9 oben). Bedingte Sprünge können durch Statusbits im CR3 Register sowie spezielle Kontrollbits (Branch Input B oder Address Extenderbits XB, wenn mehrere 10801 parallel benutzt werden) erzeugt werden.

Subroutinen können entweder einmal oder wiederholt durchlaufen werden. Im letzteren Fall wird in das Repeatregister CR1 die Anzahl der Wiederholungen eingeladen, dadurch wird die Wiederholung automatisiert.

Das CR1-Register kann auch die Mikroprogramm-Adresse speichern, falls Interrupts bearbeitet werden müssen.
Das CR2-Register enthält die Startadresse für das Mikroprogramm.
Das CR3-Register enthält normalerweise die Statusbits für bedingte Sprünge.

Die in der 10801 vorhandenen 16 Instruktionen sind in Bild 10 zusammengestellt. Man erkennt das normale Inkrementieren, die verschiedenen Sprünge zu den auf den Buseingängen stehenden Adressen, die Subroutinensprünge und Rücksprünge, sowie verschiedene Verzweigungen.

Durch geschicktes Ausnutzen dieser Instruktionen kann man Mikroprogrammspeicherraum und Programmentwicklungszeit sparen, z.B. für einen 8 bit Shift braucht man 2 Mikroprogrammworte:

- einen RSR-Befehl, um die Wiederholungszahl (8) in das CR1-Reg zu laden,
- einen RPI-Befehl, um die 8 Shifte auszuführen.

Schaltungen dieser Art bilden die Rechner Kontrolleinheit (CCU genannt). Sie sorgen dafür, daß im Mikroprogramm die richtige Instruktion geholt und an die ALU transportiert wird.
Die ALU wird unser nächstes Thema sein.

4. Arithmetische und logische Einheit

Die erste ALU, die als Slice zu haben war, ist die bekannte 74181, die zwei Operanden je 4 bit arithmetisch und logisch behandeln kann. Die 16 verschiedenen Funktionen werden über Mode- und Funktionsselektleitungen programmiert, die Statusein- und -ausgabesignale werden ebenso wie das Ergebnis der Operation ausgegeben.
Die ALU ist in 4 bit-Schritten kaskadierbar. Bild 11 zeigt den Baustein.

Um daraus einen effektiven Prozessor zu machen, müssen verschiedene Elemente hinzugefügt werden. Z.B. die in Bild 12 gezeigten Daten Latchregister, die die Operanden halten. Der erste Operand wird in Register A geladen, die Operation F = A ausgeführt und das Ergebnis, d.h. der erste Operand in Register B gelatcht. Der zweite Operand wird dann in Register A gelatcht, eine arithmetische oder logische Operation zwischen A und B kann ausgeführt werden. Das Ergebnis kann in Register B gespeichert werden oder über F ausgegeben werden.

Das Registerkonzept kann man ein bißchen ausbauen (Bild 13) an Stelle der Einzelregister wird ein 2-Port-RAM eingesetzt, zu dem 2 Adressen, nämlich A und B, gleichzeitig zugreifen können. A und B sind 4 bit Adressen, da es 16 Register, also 16 RAM Worte gibt.
Das RAM hat nur einen WRITE-Port, d.h. nur über die Adresse B kann geschrieben werden.

Wenn wir diesen Registersatz mit der ALU verbinden, ergibt sich Bild 14.
Die ALU-Operanden können aus A und B gleichzeitig gelesen werden, das Ergebnis wird entweder ausgegeben oder nach B zurückgeschrieben.

Shift und Rotatefunktionen sind in einer guten CPU wichtige Operationen, deswegen werden wir unserem System ein Shiftnetzwerk zufügen (Bild 15), das sowohl links schieben
rechts "
als auch nicht "
erlaubt. Die Shiftoperation wird durch 2 weitere Kontrolleitungen bestimmt, durch die bidirektionalen Ein- bzw. Ausgänge werden die Stufen kaskadierbar.

Schließlich erweitern wir noch das Eingabefeld der ALU über einen Daten-MUX (Bild 16), so daß wir statt bisher 2 nun 5 Datenquellen an die ALU legen können, nämlich

- das A-Register
- das B-Register
- einen logisch 0-Operanden
- einen D-Eingang, der Daten von außen in das System lädt und
- einen Q-Operanden, über den wir gleich noch sprechen.

Nun können wir auch einen Ausgangs MUX zufügen, der entweder das ALU-Ergebnis ausgibt oder den Inhalt des durch A adressierten Registers.

Die letzte Erweiterung ist in Bild 17 zu sehen. Es wird ein Q-Register eingeführt, das für Multiplikations- und Divisionsroutinen verwendet wird und bei längeren Produkten bzw. Quotienten mit dem RAM-Register in Serie geschaltet werden kann. Dafür muß das Q-Register natürlich sein eigenes Shiftnetzwerk erhalten, das mit dem RAM Shift in Serie geschaltet werden kann.

Nun haben wir ein System geschaffen, das der 4 bit ALU weit überlegen ist, ein komplettes CPU-System, das in 4 bit-Slices als 2900-System von verschiedenen Herstellern angeboten wird. (Bild 18)
Es kann die Source Operanden aus 5 verschiedenen Quellen holen, es kann 8 verschiedene Operationen ausführen, nämlich 3 arithmetische und 5 logische, es kann das Ergebnis entweder ausgeben oder es geshiftet oder ungeshiftet in das Q-Register oder in das RAM einladen.
Die verschiedenen Operationen, die Bild 19 zeigt, werden über 9 Mikrocode-Leitungen gesteuert, davon werden

- 3 für die Sourceoperanden Kontrolle,
- 3 für die ALU Funktionen,
- 3 für die Destination Kontrolle

benutzt.
Das 2900-System ist in Schottky-TTL aufgebaut, es erreicht Befehlsausführungszeiten von weniger als 200 ns.

Wenn wir kürzere Operationsabläufe erreichen wollen, müssen wir zur ECL-Technik übergehen.
Das System 10800 ist nicht nur schneller, sondern auch leistungsfähiger und komplexer als das 2900-System.
Es hat ein 3-Bussystem, wo

- der Ø-Bus und der I-Bus bidirektional sind, um Daten zu oder von der ALU zu transportieren sowie einen
- A-Bus, der nur ein Eingangsbus ist.

Die im System auftauchenden

AS pins (AS0 bis AS16) sind Source, Destination bzw. Instruktions Selektierungspins

AC bzw. LC pins sind Clock inputs für das Latchen der Daten.

Wir können vom Blockbild des 2900-Systems zu dem des 10800-Systems kommen, wenn wir einiges umordnen. (Bild 20 oben).
Zunächst wird der Registersatz, das 16-Worte RAM aus der CPU genommen, um dem Entwickler die Freiheit zu geben, ein beliebiges ECL-RAM dort einzusetzen.

Der zweite Unterschied liegt im Bussystem. Sowohl der 2900-Prozessor als auch der 10800-Prozessor verwenden 3 Busse, beim 10800 ist aber der I-Bus und der O-Bus bidirektional und nicht unidirektional wie beim 2900. Die Daten können zur oder aus der ALU transportiert werden.
Das mittlere Bild 20 zeigt einen weiteren Unterschied. Das Q-Register und der Shifter sind durch einen Akkumulator ersetzt, er ist das einzige Register in der CPU, das geclockt werden muß.
Das untere Bild 20 zeigt, daß das Shiftnetzwerk nach der ALU eingebaut ist. Alle Daten, die durch die ALU gehen, können dann geshiftet werden. Aber auch Daten aus dem Akkumulator können über das Shiftnetzwerk gehen.

Wenn wir jetzt noch Kontrollschaltungen für den I-Bus, den O-Bus und eine Maske für das Datenlatch einfügen, ist die Transformation in das 10800 System schon zu Ende. Bild 21 zeigt den Gesamtaufbau.

Unser nächstes Bild 22 zeigt das komplette Diagramm des 10800-Prozessors, der als 4 bit-Slice auf dem Markt ist.

Die 17 Mikroprogrammsteuerleitungen sind:

AS0, AS1 kontrollieren die Datenquelle des Y_A-Eingangsfeldes

AS2, AS3 kontrollieren die Datenquelle des X_{in}-Eingangsfeldes

AS4 kontrollieren ein INCR oder DECR der ALU um 2.

AS5, AS6 kontrollieren die Destination des Akkumulatorausgangs

AS7 kontrollieren die Informationsquelle des Shiftnetzwerkes

AS8 enabled oder disabled die I-Bustreiber nach außen

AS9 s. AS15

AS10, AS11 bestimmen den Arithmetischen Mode (ADD, SUB, BCD, BIN)

AS12 bestimmt, ob ADD=ADD oder =XOR (Logic), bei Logic ohne Carry

AS13, AS14 kontrollieren das Shiftnetzwerk

AS15 zusammen mit AS9 kontrollieren 2 Funktionen, nämlich

1. die Datenquelle für den Akkumulator

2. die Datenquelle für den I-Bus Treiber

AS16 ist für Latch Clock Enable oder Disable

R1/R4 sind die Shiftverbindungen,

C_{in}/C_{out} die Carry verbinder,

Group Propagate, Group Generate PG, GG für Carry look ahead Signale,

OVF als Status aus der ALU zeigt arithmetischen Overflow an,

ZERO Detect zeigt nur Nullen im Shiftnetzwerk,

P_c wird bei Parity Check benutzt.

Der 10 8000 ist in einem 48 pin QUIL-Gehäuse untergebracht.

Die Arbeitsgeschwindigkeit der beiden als 16 bit geschalteten Slice-Prozessoren 2901 und 10800 ist in Bild 23 dargestellt, wobei für beide das Carry Look Ahead-Obertragsverfahren angenommen wurde, d.h. die Carry Outs, Group Propagate, Group Generate-Signale werden Sliceweise an den Look ahead Baustein übertragen.

Dann ergibt sich für ein 16 bit - 2901/2902 System (Bild 23 oben):

- Von den Adresseingängen des RAM bis zum Carry Out eine Zeit von 120.5 ns
- Von den Adresseingängen des RAM bis zum Data Out eine Zeit von 145.5 ns.

Für eine 16 bit - 10800/10179 System (Bild 23 unten) erhält man:

- Von den A- und Ø Bus Eingänge bis zum Carry Out eine Zeit von 29.5 ns
- Von den A- und Ø Bus Eingängen bis zum I-Bus Ausgang eine Zeit von 41 ns.

Die ECL-Zeiten sind aber nicht ganz vergleichbar; weil das RAM nicht mit eingebaut ist. Wählt man z.B. das 10143 Multiport Register, müssen 16 ns hinzugerechnet werden, bis die Daten an den A- bzw. Ø Buseingängen stehen, so daß sich statt 41 ns jetzt 57 ns ergeben.

Wenn man die Befehlszyklusarten abschätzen will, muß man für den Non Pipelinig-Fall die Durchlaufzeiten durch den Sequenzer und den Mikroprogrammspeicher addieren, im Pipelining-Fall nur die Laufzeit durch das Pipelineregister.

Für das 2900-System ist die Durchlaufzeit in Bild 24 für beide Fälle gezeichnet.

Man kommt auf etwa 320 ns für Non-Pipelining und etwa 180 ns mit Pipelining.

In den Bildern 25 und 26 sind die Angaben für das 10800 System; sie sind 87 ns bzw. 61 ns und damit etwa 3 mal so schnell wie im 2900-System.

5. Das I/O- oder Memory Interface

Zu Anfang hatte ich gesagt, daß die Sliceprozessoren in 2 verschiedenen Betriebsarten eingesetzt werden, als Kontroller oder als Emulator.

Als Kontroller enthalten sie ihr gesamtes Programm im Mikroprogrammspeicher, als Emulator haben sie im externen Speicher Instruktionen stehen, die den Maschineninstruktionen eines größeren Rechners entsprechen, z.B. einer IBM. Für den Mikroprozessor sind es Makrobefehle, die er nacheinander in das Instruktionsregister des Mikroprogrammsequenzers holt, dort decodiert und in eine Anzahl von Mikroschritten auflöst und ausführt. In diesem Fall hat der Prozessor 2 Instruktionsspeicher, einen für das Makroprogramm, einen für das Mikroprogramm. Da in beiden Programmen die nächste Adresse durch Inkrementieren, Springen, Verzweigen oder Subroutinensprünge gefunden werden muß, muß eine Art Makrosequenzer vorhanden sein, der ähnlich wie der Mikrosequenzer aufgebaut sein wird. In beiden erwähnten Systemen gibt es diese Bausteine, im TTL-System heißt er 2930, im ECL-System 10803. Sie sollen hier kurz vorgestellt werden.

Bild 27 zeigt den 2930. Er ist eine 4 bit breite Programmkontroll Einheit, Kaskadierbar, so daß 4 Bausteine eine 16 bit Adresse erzeugen, d.h. 64 K direkt adressieren können.
Der 2930 enthält:

- einen Volladdierer mit Eingangs-MUX,
- einen Programmzähler PC mit Incrementer und Eingangs-MUX,
- ein 17 x 4 LIFO Stack mit Eingangs-MUX und Stackpointer,
- ein Hilfsregister mit Eingangs-MUX und
- einen Instruktionsdecoder, der 32 verschiedene Instruktionen decodiert.

Es gibt 5 Klassen von Instruktionen:

- Unbedingtes Holen,
- Bedingte Sprünge
- Bedingte Sprünge zu Subroutinen
- Bedingte Sprünge von Subroutinen
- Verschiedenes

Der Volladdierer, der die relativen Adressen errechnet, kann die Operanden dazu aus 4 Quellen holen:

- vom PC
- vom Stack
- vom Hilfsregister, das Indexwerte oder zwischengespeicherte Adressen enthält
- vom direkten Eingang

Die Schaltung ist im wesentlichen eine Kombination des besprochenen Mikroprogramm-Sequenzers und eines Addierers.
Der entsprechende ECL-Baustein 10803 ist in Bild 28 gezeigt.
Ein Speicherdatenregister MDR hält die einkommenden oder ausgehenden Daten, ein Speicheradressregister MAR die ausgehende Adresse. Ein 4 x 4 Register File kann als PC, als Stackpointer, als Indexregister oder ähnliches verwendet werden. Der File Ausgang geht entweder in die ALU oder in das MAR.
Ein Datenmatrix-Block kontrolliert den Datentransfer zwischen verschiedenen internen Registern und den I/O-Ports. 17 verschiedene Datentransfer Operationen sind möglich, sie werden durch die MS-Signale kontrolliert.
Die MS 14 Leitung erlaubt durch Invertieren der Daten auf dem Datenbus und dem Addressbus das Verwenden negativer oder positiver Logik. Einmal selektiert, wird jeder Transfer in einem Mikrozyklus ablaufen.

Die ALU kann ADD, SUB, Shift Left, Shift Right, AND, OR und XOR Funktionen ausführen, um die relativen oder indizierten oder erweiterten Adressen zu bilden und Stackpointeroperationen ausführen.
Sie holt ihre Operanden über MUX vom I- oder Ø-Bus, dem MDR, dem Register File, dem PC, dem MAR oder den Pointereingängen, die Adressenoffsets addieren können bzw. Increments, Decrements der Adresse oder Bitmasken erzeugen.

Die ALU und die Datarouting Matrix werden getrennt gesteuert, ihre Operationen können während eines Mikrozyklus parallel ablaufen.

6. Software-Entwicklung für Slice Prozessoren

Die Auslegung von verschiedenen Benützersystemen variert sehr stark je nach der Anwendung (Kontroller, Emulator etc.) Aus diesem Grunde ist es sehr schwierig, einen allgemeinen Assembler, Editor und Simulator für alle möglichen Anwendungen zu konzipieren. Andererseits ist es sehr mühsam, wenn nicht gar unmöglich, ein Mikroprogramm eines gewissen Umfanges ohne jedes Hilfsmittel zu entwickeln und dessen Fehler zu bereinigen. Motorola hat deshalb ein spezielles Systementwicklungsgerät entwickelt, das erstens das Schreiben von Mikroprogrammen und zweitens das Fehlerbereinigung dieses Mikroprogrammes ganz wesentlich erleichtert.

Um das Systementwicklungsgerät einsetzen zu können, muß zuerst das Benützersystem konzipiert und entwickelt werden. Dann wird der Mikroprogrammspeicher dieses Systems herausgenommen oder abgeschaltet und durch den kompletten Exorciser mit seinen Peripheriegeräten ersetzt (Bild 29). Der Exorciser figuriert nun als ein "Prom Simulator". Diese Besonderheit erlaubt es, Mikrobefehle oder Worte (in hexadecimaler Form) zu schreiben, die Worte beliebig zu ändern, dann das ganze Mikroprogramm auf Band oder Diskette zu speichern und später wieder in den Hauptspeicher zu laden.

In einem weiteren Schritt kann das speziell entwickelte Softwareprogramm, FAST genannt, benutzt werden, um das Mikroprogramm Schritt um Schritt oder mit reduzierter Geschwindigkeit ablaufen zu lassen und eventueller Fehler zu korrigieren. Zum Schluss kann das ganze (oder auch nur ein Teil) des Mikroprogrammes in einen schnellen ECL-RAM Speicher geladen werden (RAML) und in Echtzeit ausgeführt werden (REAL). Beim Antreffen von "Breakpoints" wird das Programm angehalten und die Zustände der verschiedenen Busse ausgedruckt.

FAST-Befehle beschreiben:

- Definition der Hardware Konfiguration,
- Weglegen und Heraussuchen von Benutzerdateien,
- Mikroprogramm-Ausführung und Austesten im MOS RAM,
- Realtime Ausführung und Austesten im ECL-RAM.

Der gesamte Ablauf ist in den 5 Bildern 30-34 dargestellt.

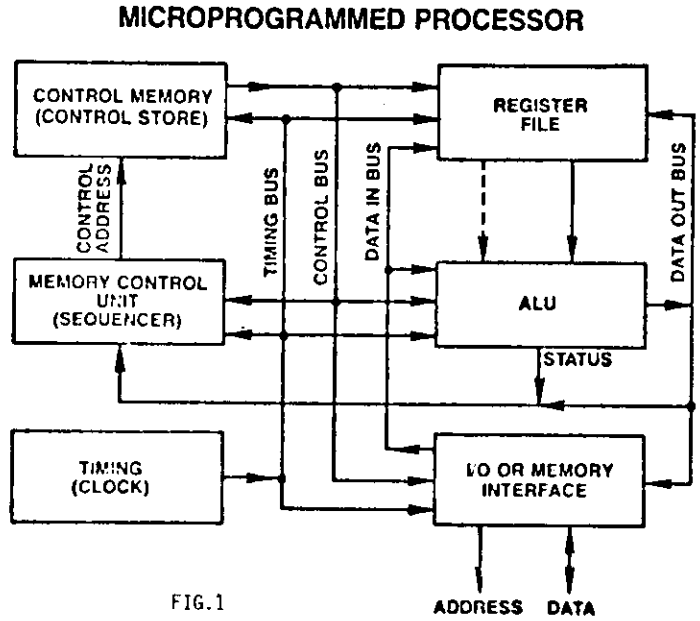


FIG.1

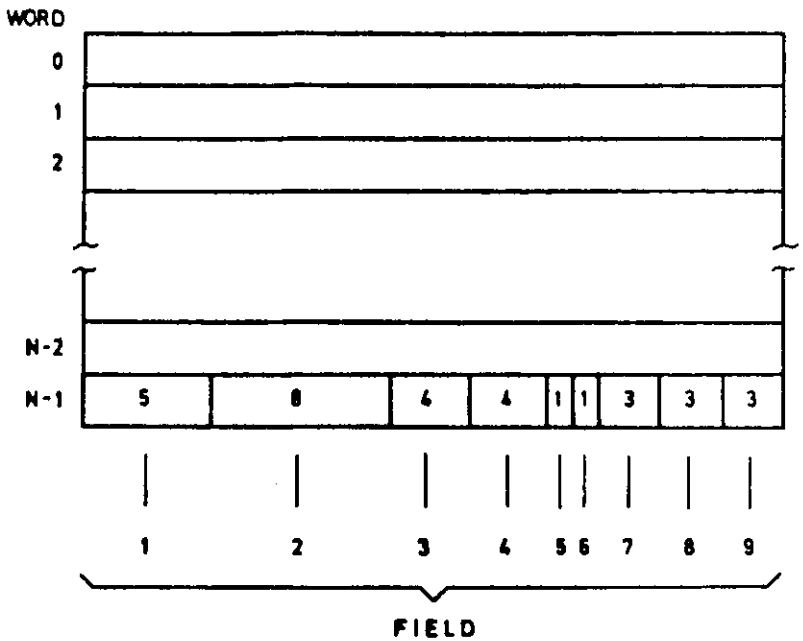

WORD
0
1
2
N-2
N-1
5
8
4
4
1
1
3
3
3
1
2
3
4
5
6
7
8
9
FIELD

FIG.2

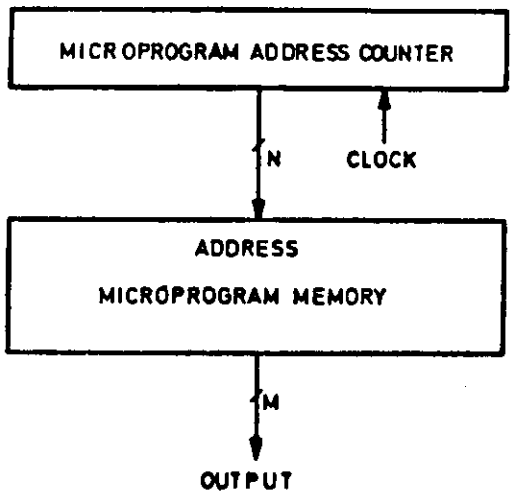

MICROPROGRAM ADDRESS COUNTER
N
CLOCK
ADDRESS
MICROPROGRAM MEMORY
M
OUTPUT

FIG.3

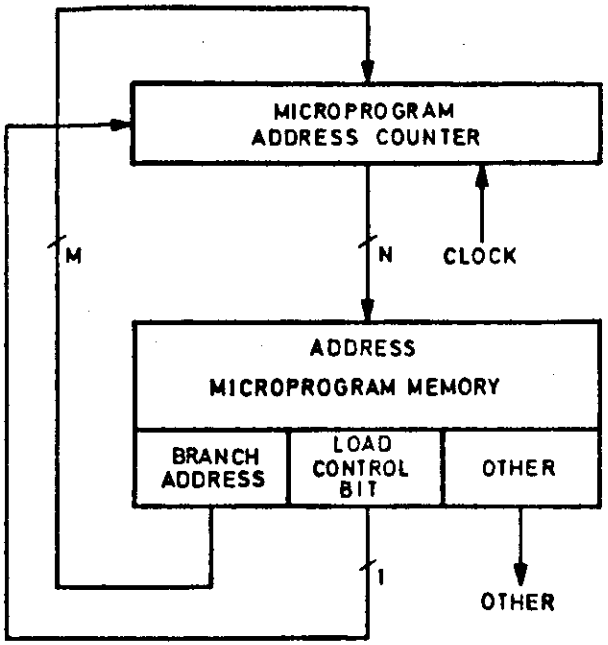


FIG.4

S_1	S_2	FUNCTION
0	0	CONTINUE
0	1	JUMP IF CONDITION 1 IS TRUE
1	0	JUMP IF CONDITION 2 IS TRUE
1	1	JUMP UNCONDITIONAL

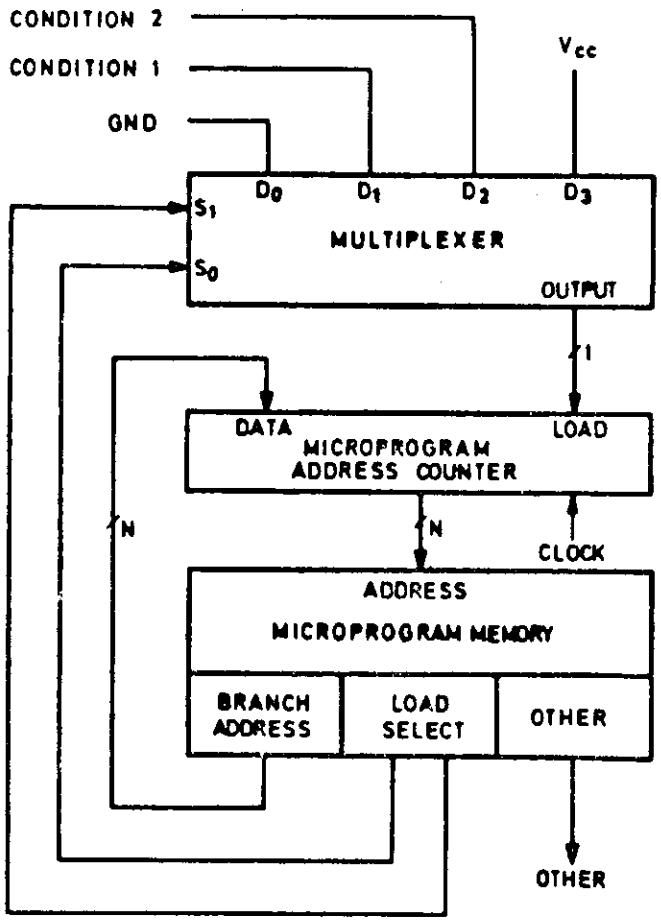


FIG.5

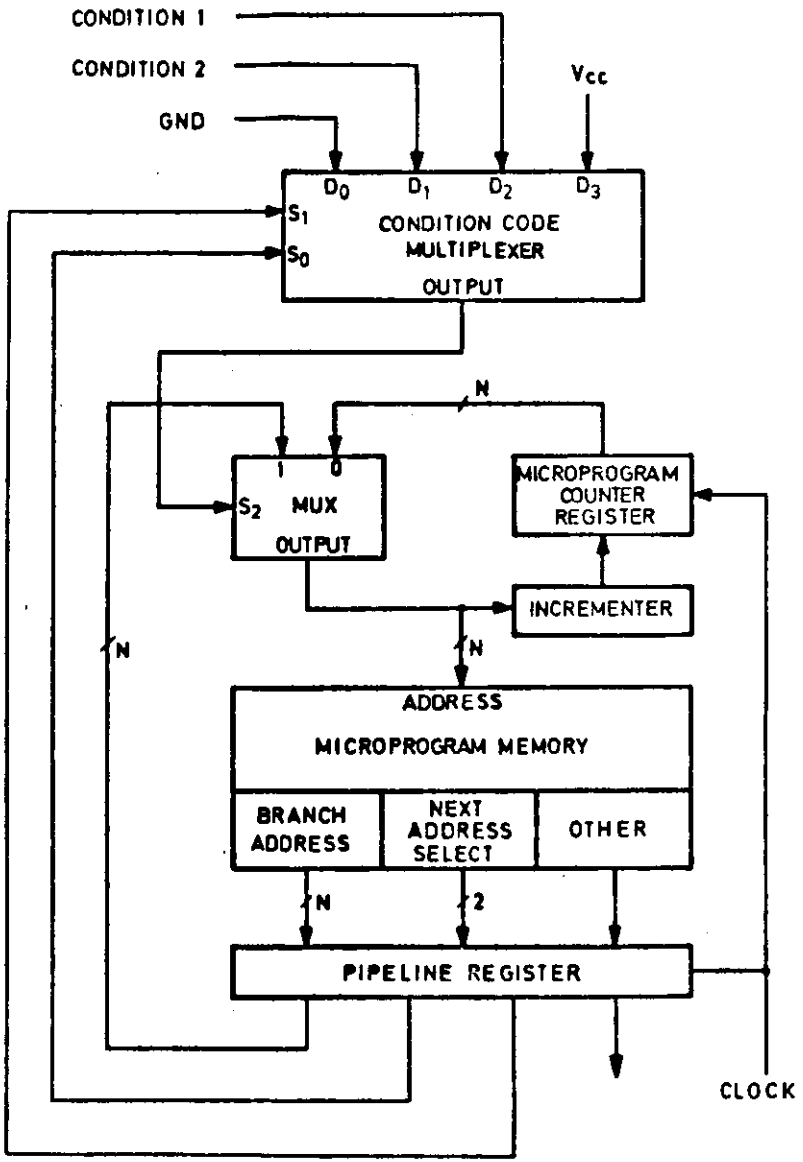

CONDITION 1
CONDITION 2
GND
Vcc
D0 D1 D2 D3
S1
S0
CONDITION CODE MULTIPLEXER
OUTPUT
N
1 0
S2 MUX
OUTPUT
MICROPROGRAM COUNTER REGISTER
INCREMENTER
N
N
ADDRESS
MICROPROGRAM MEMORY
BRANCH ADDRESS
NEXT ADDRESS SELECT
OTHER
N
2
PIPELINE REGISTER
CLOCK

FIG.6

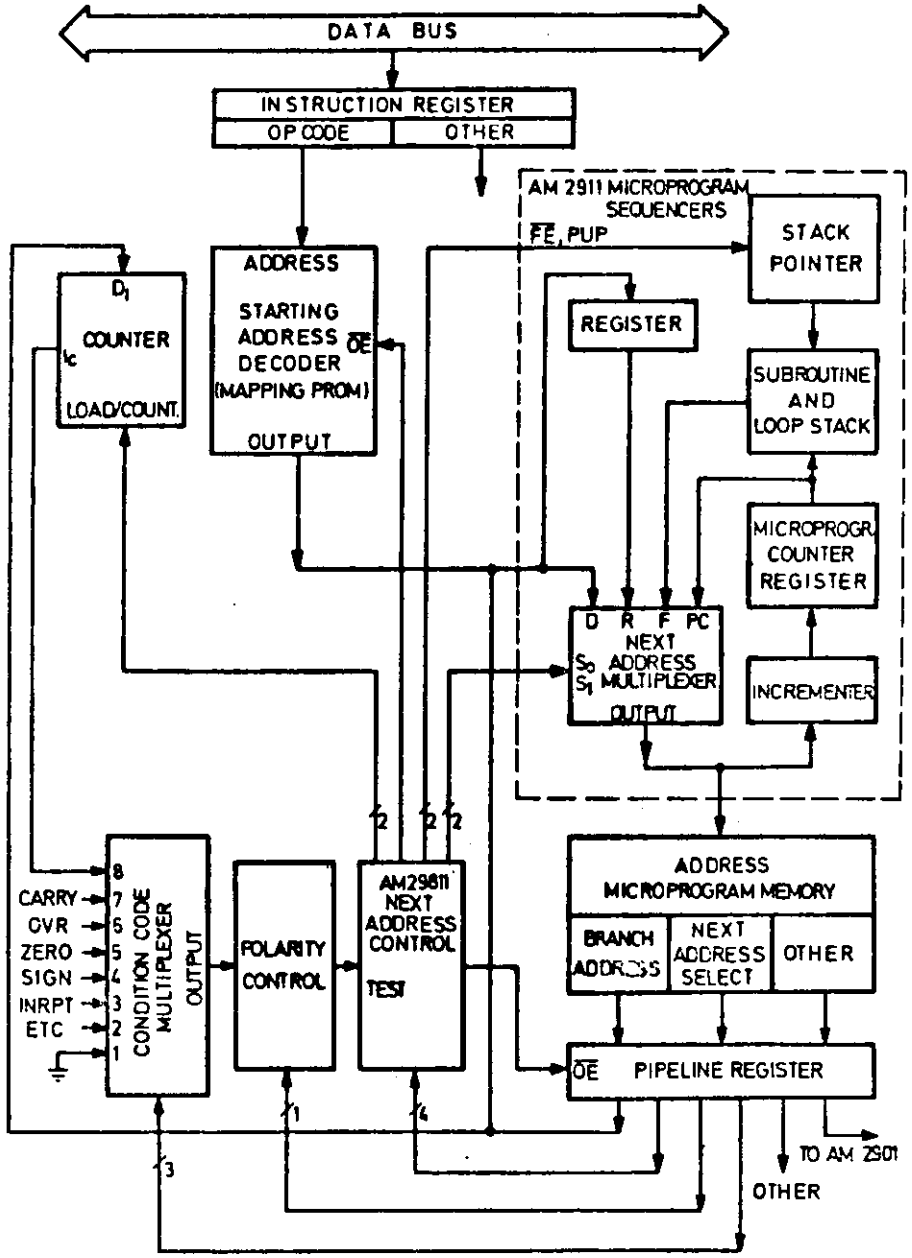
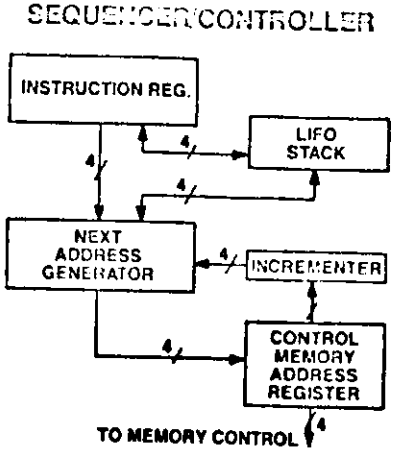
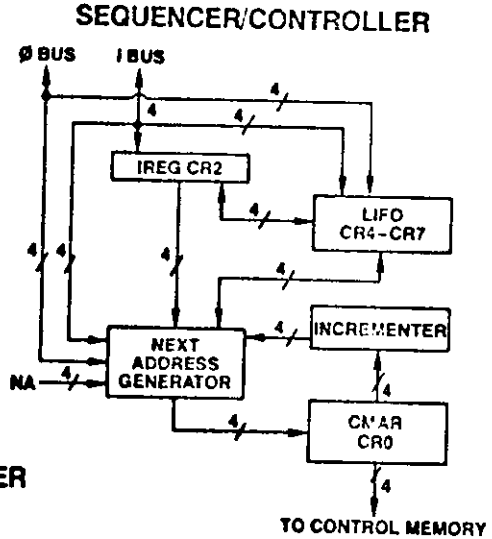

DATA BUS
INSTRUCTION REGISTER
OP CODE
OTHER
AM 2911 MICROPROGRAM SEQUENCERS
FE, PUP
STACK POINTER
REGISTER
SUBROUTINE AND LOOP STACK
MICROPROGR. COUNTER REGISTER
D1
COUNTER
LOAD/COUNT.
ADDRESS
STARTING ADDRESS DECODER (MAPPING PROM)
OE
OUTPUT
D R F PC
NEXT ADDRESS MULTIPLEXER
OUTPUT
INCREMENTER
2 2 2
CARRY 8
OVR 7
ZERO 6
SIGN 5
INRPT 4
ETC 3
2
1
CONDITION CODE MULTIPLEXER OUTPUT
POLARITY CONTROL
AM29811 NEXT ADDRESS CONTROL
TEST
ADDRESS MICROPROGRAM MEMORY
BRANCH ADDRESS
NEXT ADDRESS SELECT
OTHER
OE PIPELINE REGISTER
1
4
3
TO AM 2901
OTHER

FIG.7


SEQUENCER/CONTROLLER
INSTRUCTION REG.
LIFO STACK
NEXT ADDRESS GENERATOR
INCREMENTER
CONTROL MEMORY ADDRESS REGISTER
TO MEMORY CONTROL


SEQUENCER/CONTROLLER
Ø BUS
I BUS
IREG CR2
LIFO CR4-CR7
INCREMENTER
NEXT ADDRESS GENERATOR
NA
CMAR CR0
TO CONTROL MEMORY

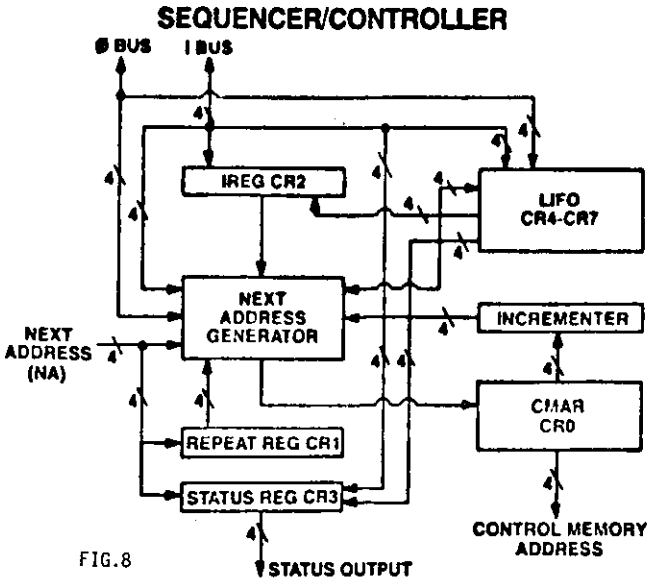

SEQUENCER/CONTROLLER
Ø BUS
I BUS
IREG CR2
LIFO CR4-CR7
NEXT ADDRESS GENERATOR
NEXT ADDRESS (NA)
INCREMENTER
CMAR CR0
REPEAT REG CR1
STATUS REG CR3
CONTROL MEMORY ADDRESS
STATUS OUTPUT

FIG.8

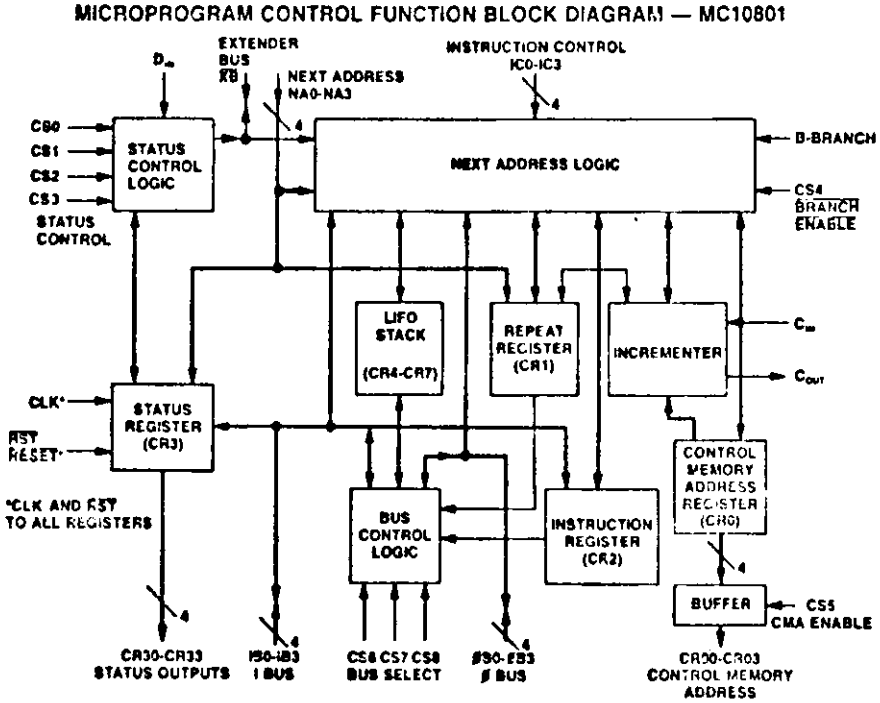

MICROPROGRAM CONTROL FUNCTION BLOCK DIAGRAM — MC10801
EXTENDER BUS
INSTRUCTION CONTROL IC0-IC3
NEXT ADDRESS NA0-NA3
CS0
CS1
CS2
CS3
STATUS CONTROL
STATUS CONTROL LOGIC
NEXT ADDRESS LOGIC
B-BRANCH
CS4 BRANCH ENABLE
LIFO STACK (CR4-CR7)
REPEAT REGISTER (CR1)
INCREMENTER
CLK*
RST RESET*
STATUS REGISTER (CR3)
*CLK AND RST TO ALL REGISTERS
BUS CONTROL LOGIC
INSTRUCTION REGISTER (CR2)
CONTROL MEMORY ADDRESS REGISTER (CR0)
BUFFER
CS5 CMA ENABLE
CR30-CR33 STATUS OUTPUTS
I30-I33 I BUS
CS6 CS7 CS8 BUS SELECT
Ø30-Ø33 Ø BUS
CR30-CR03 CONTROL MEMORY ADDRESS

FIG.9

INC – Increment
JMP – Jump to NA Inputs
JIB – Jump to I Bus
JIN – Jump to I Bus and Load R2
JPI – Jump to R2
JEP – Jump to External Bus (Ø Bus)
JL2 – Jump to NA Inputs and Load R2
JLA – Jump to NA Inputs and Load R1
JSR – Jump to Subroutine
RTN – Return from Subroutine
RSR – Repeat Subroutine (Load R1 from NA Inputs)
RPI – Repeat Instruction (Jump to R1)
BRC – Branch to NA Inputs on Condition
Otherwise Increment
BSR – Branch to Subroutine on Condition
Otherwise Increment
ROC – Return from Subroutine on Condition or jump to NA Inputs
BRM – Branch and Modify address with Branch inputs (multiway Branches)

FIG.10

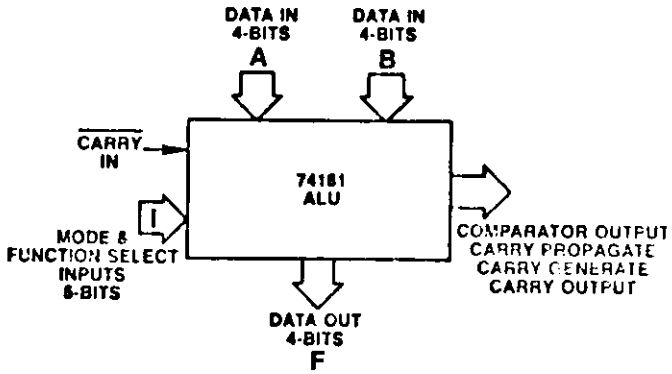


FIG.11

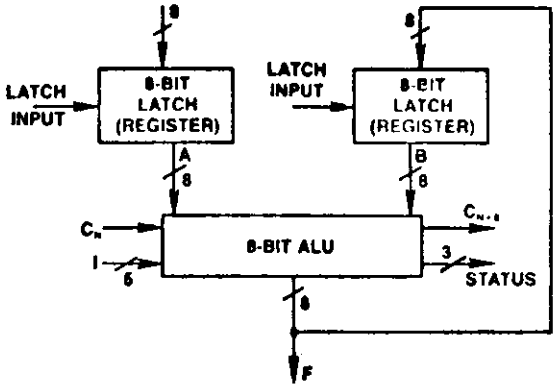

8
LATCH INPUT
8-BIT LATCH (REGISTER)
8
LATCH INPUT
8-BIT LATCH (REGISTER)
A
8
B
8
C_n
I
5
8-BIT ALU
C_{n+8}
3
STATUS
8
F

FIG.12

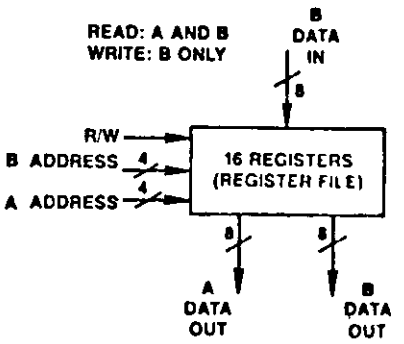

READ: A AND B
WRITE: B ONLY
B DATA IN
8
R/W
B ADDRESS
4
A ADDRESS
4
16 REGISTERS (REGISTER FILE)
8
8
A DATA OUT
B DATA OUT

FIG.13

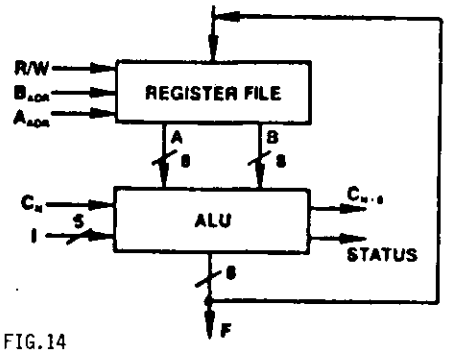

R/W
REGISTER FILE
A
B
8
8
ALU
5
STATUS
8
F

FIG.14

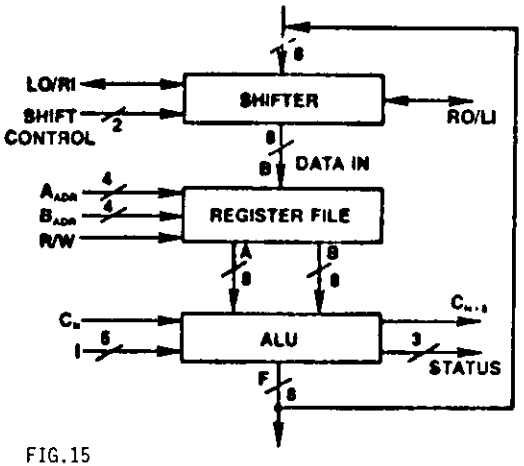

8
LO/RI
SHIFTER
RO/LI
SHIFT CONTROL
2
8
8
DATA IN
4
4
R/W
REGISTER FILE
A
B
8
8
ALU
5
3
STATUS
F
8

FIG.15

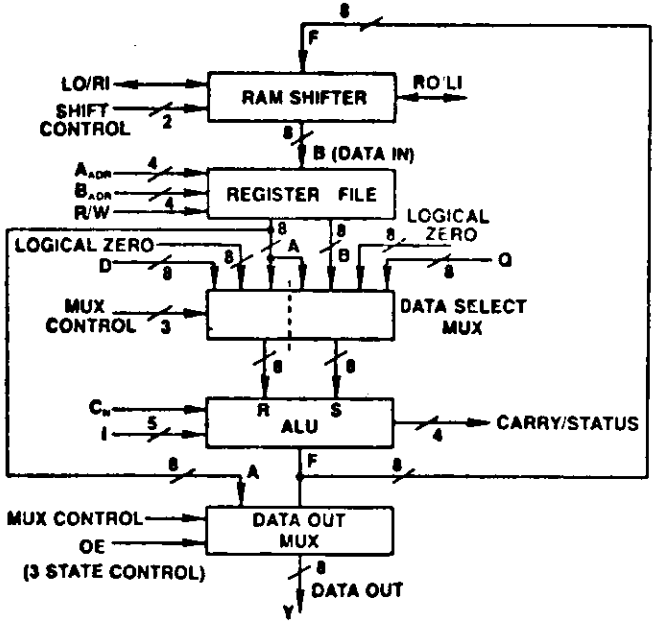

F
LO/RI
SHIFT CONTROL
RAM SHIFTER
RO/LI
B (DATA IN)
REGISTER FILE
LOGICAL ZERO
D
A
B
Q
MUX CONTROL
DATA SELECT MUX
ALU
CARRY/STATUS
DATA OUT MUX
OE
(3 STATE CONTROL)
DATA OUT

FIG.16

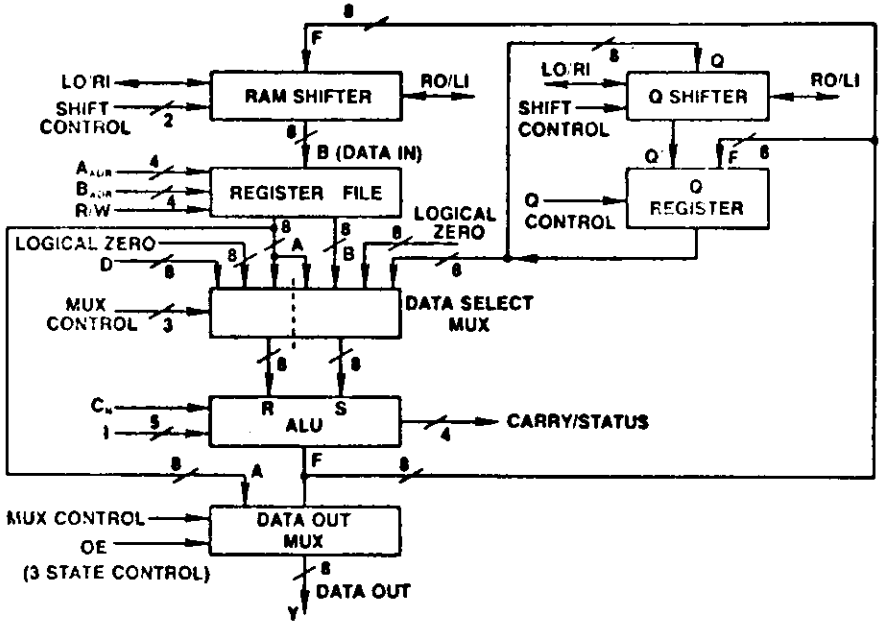

F
LO/RI
SHIFT CONTROL
RAM SHIFTER
RO/LI
Q SHIFTER
Q REGISTER
Q CONTROL
B (DATA IN)
REGISTER FILE
LOGICAL ZERO
D
A
B
MUX CONTROL
DATA SELECT MUX
ALU
CARRY/STATUS
DATA OUT MUX
OE
(3 STATE CONTROL)
DATA OUT

FIG.17

MC 2901
MICROPROCESSOR SLICE BLOCK DIAGRAM

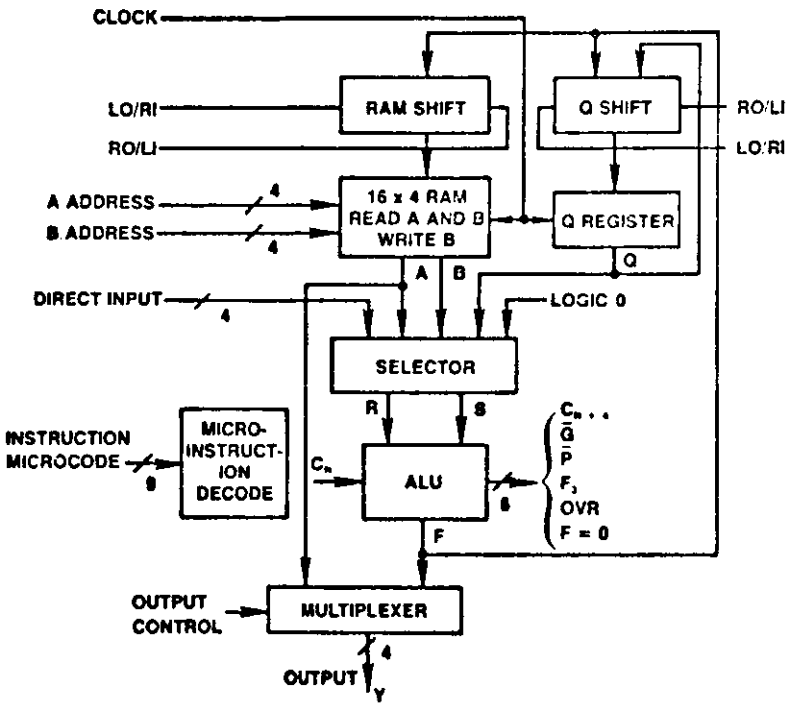


FIG.18

ALU FUNCTION CONTROL

MICROCODE I_2	I_1	I_0	ALU FUNCTION	SYMBOL
0	0	0	R PLUS S	R + S + C_n
0	0	1	S MINUS R	S - R - $\overline{C}_n$
0	1	0	R MINUS S	R - S - $\overline{C}_n$
0	1	1	R OR S	R$\vee$S
1	0	0	R AND S	R$\wedge$S
1	0	1	$\overline{R}$ AND S	$\overline{R}\wedge$S
1	1	0	R EX-OR S	R$\veebar$S
1	1	1	R EX-NOR S	$\overline{R\veebar S}$

ALU SOURCE OPERAND CONTROL

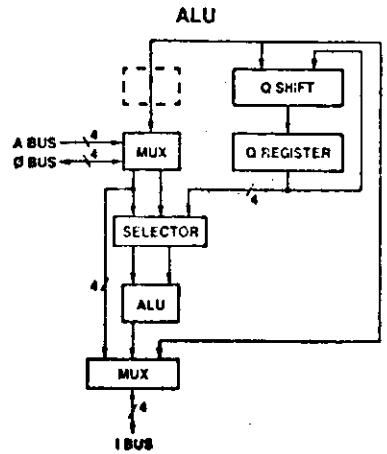
MICROCODE I_2	I_1	I_0	ALU SOURCE OPERANDS R	S
0	0	0	A	Q
0	0	1	A	B
0	1	0	0	Q
0	1	1	0	B
1	0	0	0	A
1	0	1	D	A
1	1	0	D	Q
1	1	1	D	0

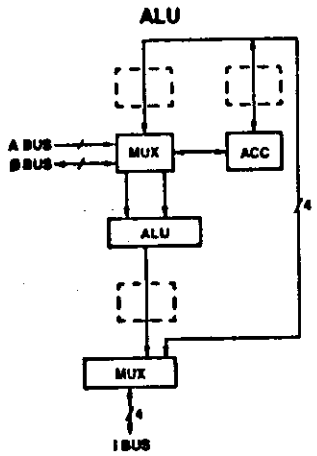
ALU OUTPUT DESTINATION CONTROL

MICROCODE I_5	I_4	I_3	LOAD	Y OUTPUT
0	0	0	F → Q	F
0	0	1	NONE	F
0	1	0	F → B	A
0	1	1	F → B	F
1	0	0	F/2 → B; Q/2 → Q	F
1	0	1	F/2 → B	F
1	1	0	2F → B; 2Q → Q	F
1	1	1	2F → B	F

B = REGISTER ADDRESSED BY "B" ADDRESS
F= ALU OUTPUT
F/2 = SHIFT RIGHT
2F = SHIFT LEFT

FIG.19





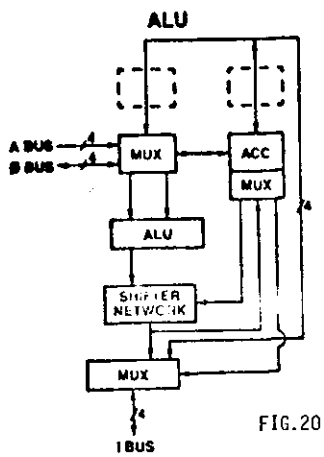


FIG.20

4-BIT ALU SLICE
— MC10800 —
DATA FLOW

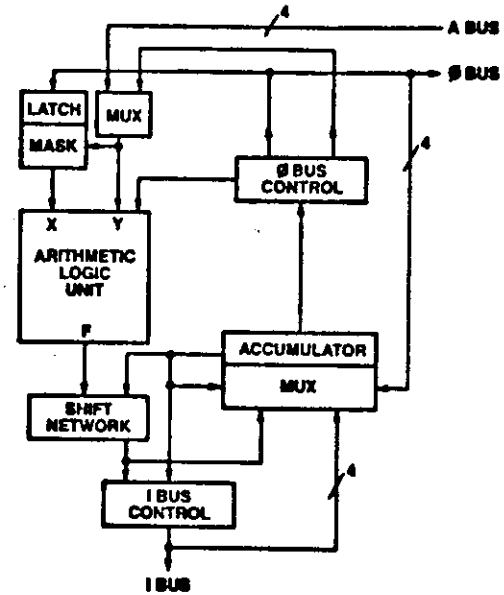


FIG.21

FUNCTIONAL BLOCK DIAGRAM

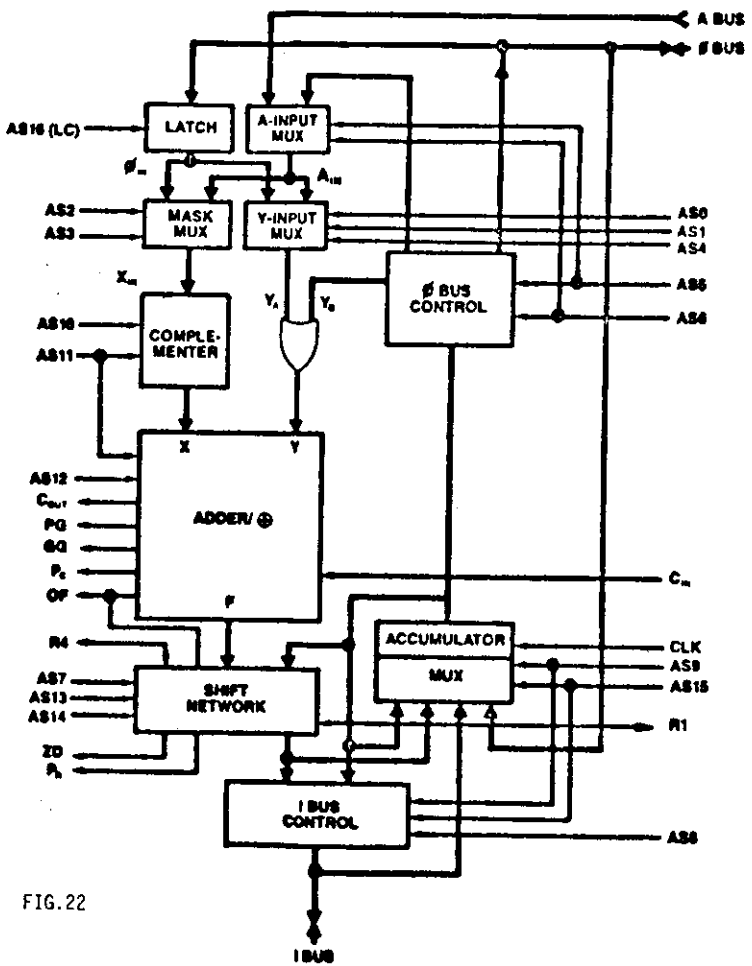
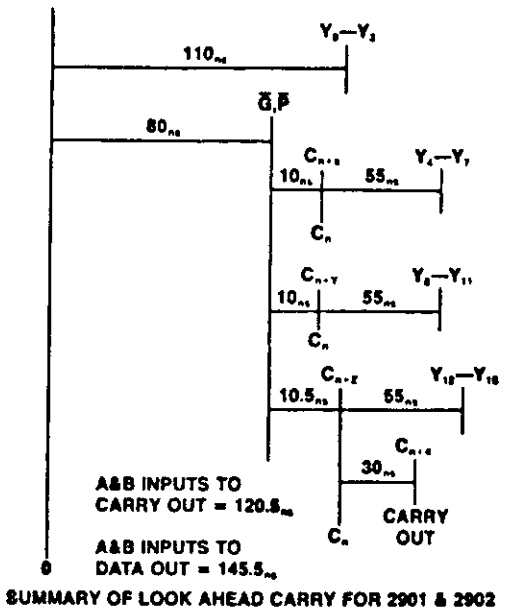


FIG.22

LOOK AHEAD CARRY



SUMMARY OF LOOK AHEAD CARRY FOR 2901 & 2902

LOOK AHEAD CARRY FOR THE 10800

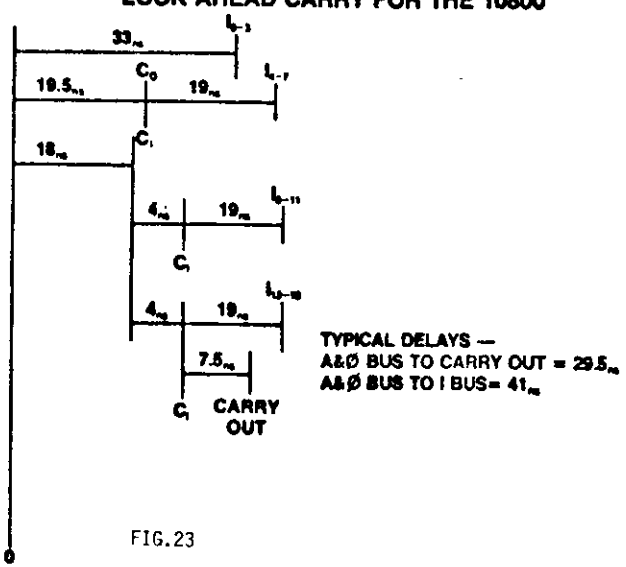


FIG.23

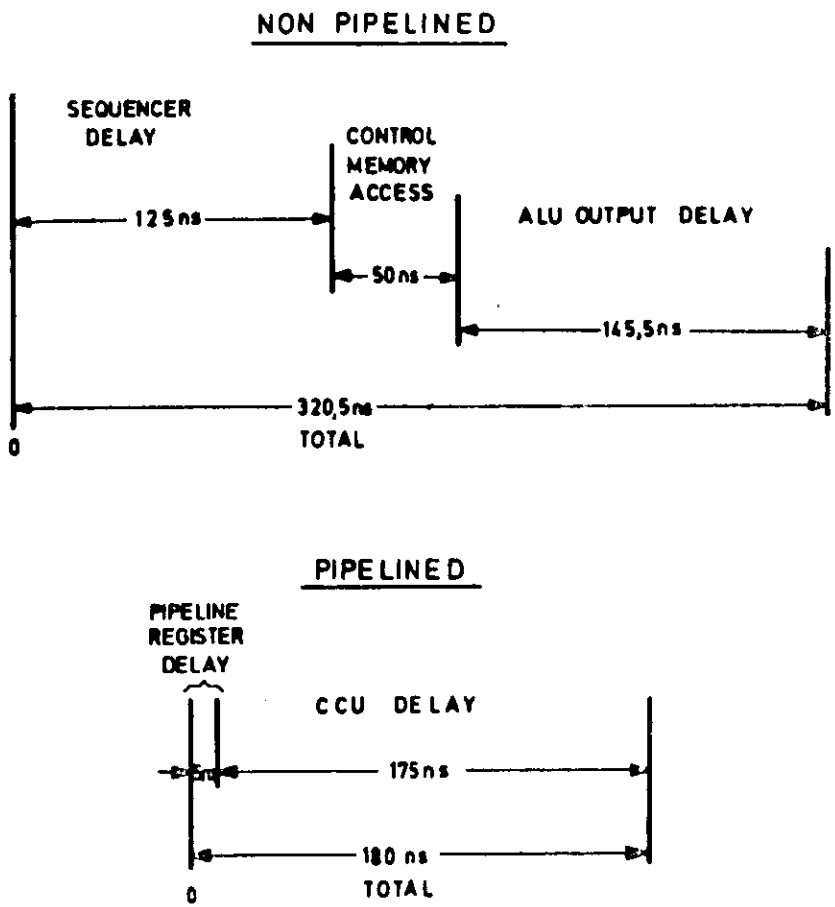

NON PIPELINED
SEQUENCER DELAY
CONTROL MEMORY ACCESS
ALU OUTPUT DELAY
125 ns
50 ns
145,5 ns
320,5 ns
TOTAL
0
PIPELINED
PIPELINE REGISTER DELAY
CCU DELAY
175 ns
180 ns
TOTAL
0

FIG.24

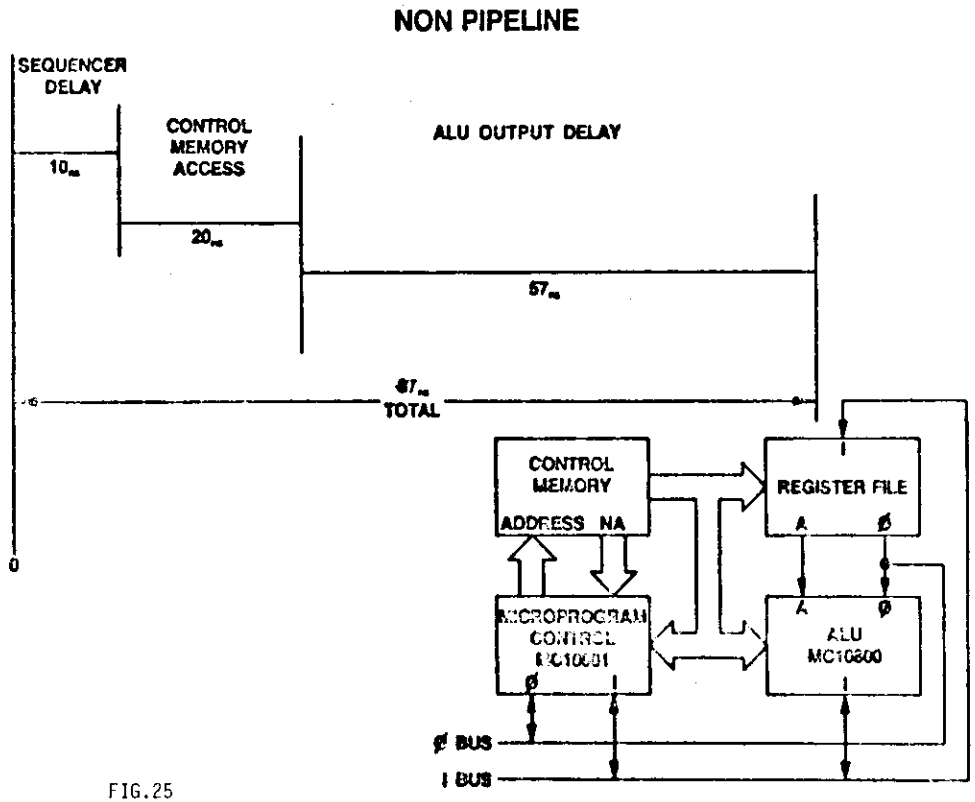

NON PIPELINE
SEQUENCER DELAY
CONTROL MEMORY ACCESS
ALU OUTPUT DELAY
TOTAL
0
CONTROL MEMORY
ADDRESS NA
REGISTER FILE
MICROPROGRAM CONTROL
ALU
MC10800
BUS
I BUS

FIG.25

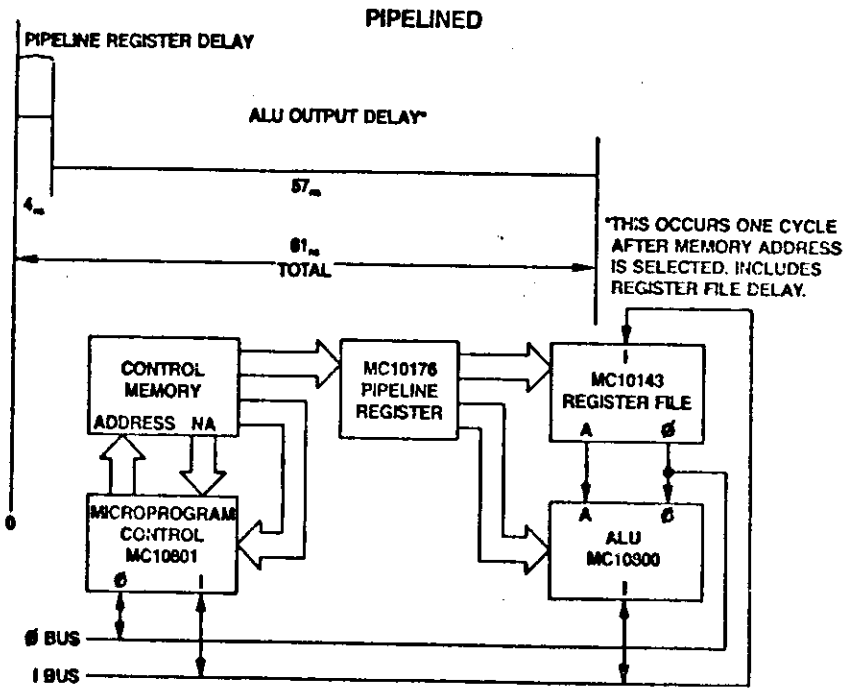

PIPELINED
PIPELINE REGISTER DELAY
ALU OUTPUT DELAY*
57ns
4ns
61ns
TOTAL
*THIS OCCURS ONE CYCLE AFTER MEMORY ADDRESS IS SELECTED. INCLUDES REGISTER FILE DELAY.
CONTROL MEMORY
ADDRESS NA
MC10176 PIPELINE REGISTER
MC10143 REGISTER FILE
A
B
MICROPROGRAM CONTROL MC10801
ALU MC10900
0
B BUS
I BUS

FIG.26

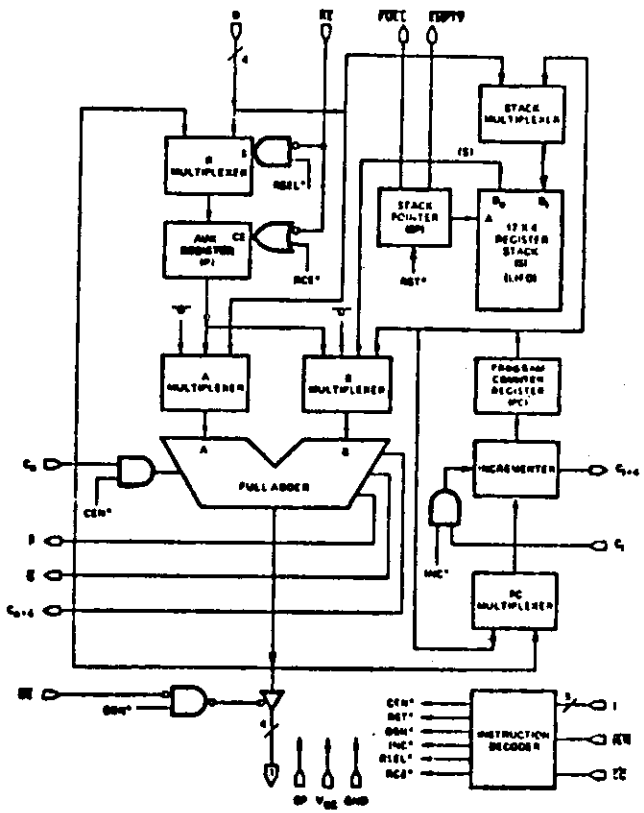

STACK MULTIPLEXER
B MULTIPLEXER
STACK POINTER
A MULTIPLEXER
B MULTIPLEXER
PROGRAM COUNTER REGISTER
FULL ADDER
INCREMENTER
PC MULTIPLEXER
INSTRUCTION DECODER

FIG.27

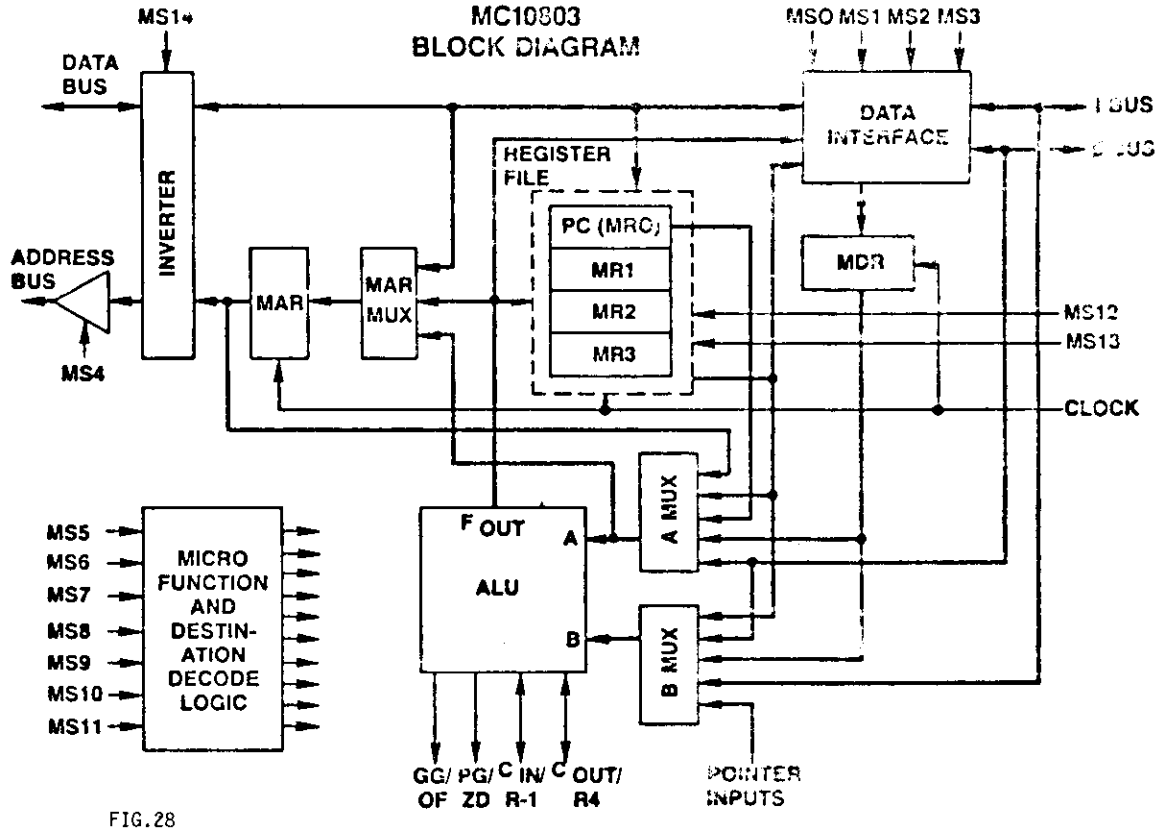

MC10803
BLOCK DIAGRAM
MS1*
DATA BUS
INVERTER
ADDRESS BUS
MS4
MAR
MAR MUX
REGISTER FILE
PC (MRC)
MR1
MR2
MR3
MSO MS1 MS2 MS3
DATA INTERFACE
MDR
MS12
MS13
CLOCK
A MUX
B MUX
F OUT
ALU
A
B
MS5
MS6
MS7
MS8
MS9
MS10
MS11
MICRO FUNCTION AND DESTIN-ATION DECODE LOGIC
GG/ OF
PG/ ZD
C IN/ R-1
C OUT/ R4
POINTER INPUTS

FIG.28

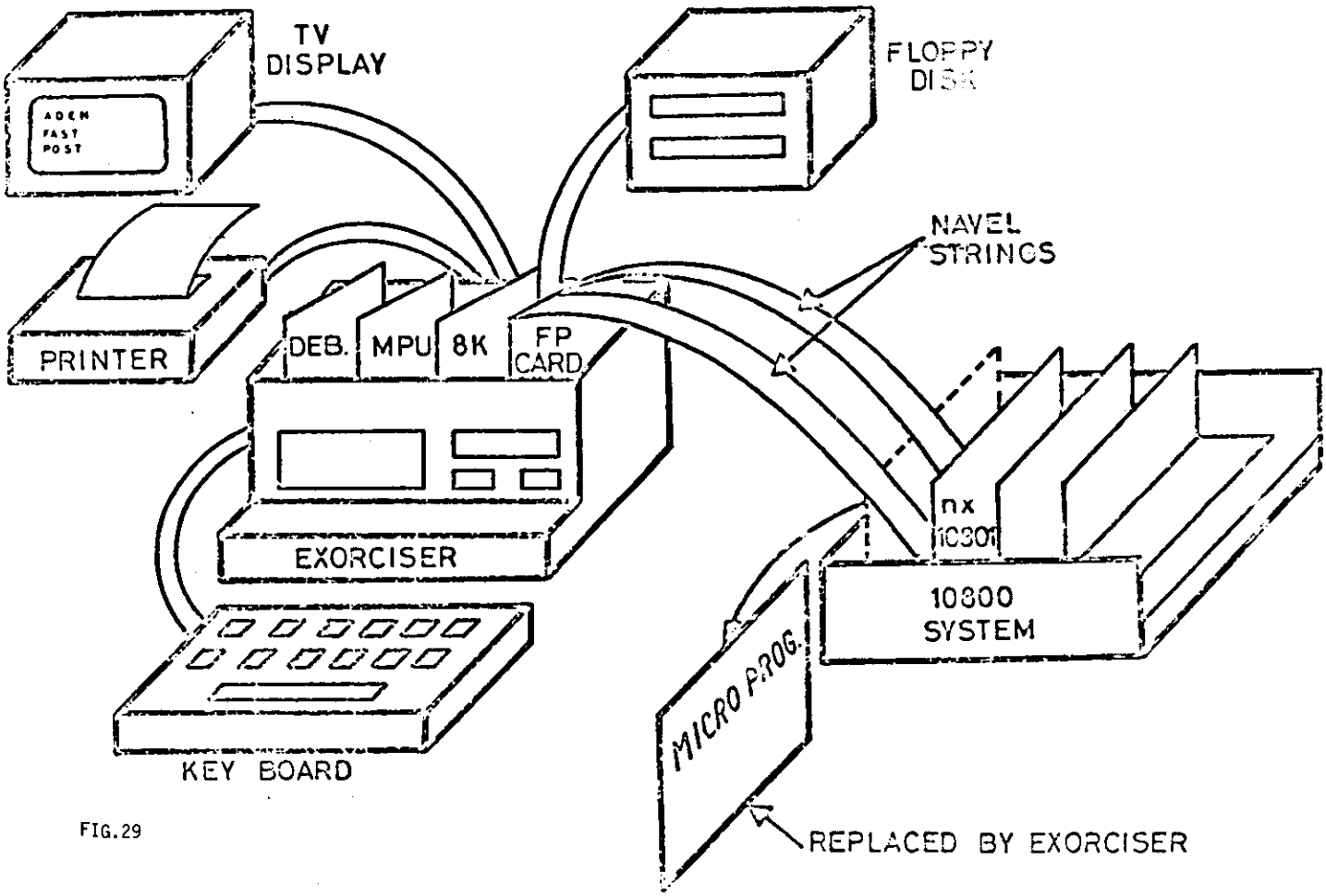

TV DISPLAY
ADEN
FAST
POST
FLOPPY DISK
NAVEL STRINGS
PRINTER
DEB.
MPU
8K
FP CARD
EXORCISER
nx 10300
10800 SYSTEM
KEY BOARD
MICRO PROG.
REPLACED BY EXORCISER

FIG.29

FUNCTIONS OF SUPPORT TOOL

1. ROM / PROM - SIMULATOR (at reduced speed)
2. RAM-LOADER (from MOS-RAM to MECL-RAM)
3. REAL-TIME EXECUTION (controlled by Exorciser)
4. PROM PROGRAMMER (from MOS RAM to PROM)

FIG.30 10800-12

ROM SIMULATOR

1st step : Build hardware with the 10800 family parts and withdraw the control memory

2nd step: Insert the cable from the Exorciser into the control memory socket

3nd step: Use the "FAST Monitor" to define the system's configuration (DECL).

4th step: Write the micro-program (in HEX) in to the MOS-RAM memory (MPGM comand of FAST)

FIG.31

5th step : Execute (simulate) and debug the micro-program step by step and at reduced speed (RUNP command of FAST)

RAM LOADER

6th step : Load the micro-program from the MOS-RAM into the MECL-RAM

FIG.32

REAL TIME EXECUTION

7th step : Execute the micro-program or part of it in real time

8th step : Correct errors using again the Rom Simulator

9th step : Iterate until all errors have been eliminated

FIG.33

PROM PROGRAMMER

10th step: Program the PROM with the help of the "PROM programmer"

11th step : Withdraw the cable to the Exorciser and insert the PROM.

FIG.34