ソフトウェア・モニタの方式設計とその応用†

宮崎正俊†† 小畑征二郎†† 松沢 茂††

計算機システムの性能を評価するための計測システムとして，NEAC シリーズ 2200 モデル 700 用に開発したソフトウェア・モニタについて，その方式設計の詳細，実測データの例，新しい応用法について述べる．

本システムはモデル 700 のオペレーティング・システムである MOD 7 のモニタの中に組込む方式を採用している．このためモニタの改造が必要となり，その制御も複雑であるが，豊富なデータを収集できる利点をもっている．データ収集の方式はサンプリングとイベントの両方式を兼ね備えており，多様な計測を行うことができる．これらの方式の制御の流れを具体的に示す．

実測例では，サンプリング方式を用いた計測として CPU とチャネルの効率を多重度を変えて示すほか，一部の周辺装置の使用率なども示す．また，イベント方式による結果として，いくつかのモニタ・モジュールの実行時間や使用頻度などを示す．

最後に，計測システムの特殊な応用として，サンプリング間隔を極めて小さくして 1 つのプログラムの実行を観測し，プログラムの動作の様子を時間とアドレス空間で表現された近似的な波形として捉えることを提案する．この波形のことをプログラム波形と呼ぶことにする．プログラム波形の実例として，単純ループを行うプログラムと FORTRAN コンパイラの場合を示す．

1. はじめに

一般に大規模な計算機システムにおいては，それが効率よく最適な状態で稼動しているかどうかを判定するのは容易なことではない．このようなシステムでは，ハードウェア構成のバランス，使用するオペレーティング・システムの能率，運用形態などが，システムの効率に大きな影響を与えることはよく知られている．

現実のシステムを対象にしてシステム評価を行う場合は，シミュレーションや数学的モデルの手法を用いるよりは，システムが稼動したときに得られるデータを直接用いる方が適切である．通常はどのシステムでも稼動記録としてある程度のデータは自動的に収集されるようにはなっているが，この種のデータだけでは一般に不十分である．つまり，稼動記録はあくまでも処理の流れの一環として記録されるものであり，システム自身の情報はあまり含まれていない．

そこで，システム動作のより詳細な情報を積極的に収集する必要が生じるわけであるが，この手法としてはハードウェア・モニタリングとソフトウェア・モニタリングの 2 つがある．ハードウェア・モニタリングはシステムの外部から電気的な信号を取り出して動作状態を観測するものである[1),2)]．ソフトウェア・モニタリングはオペレーティング・システムの中に計測するためのプログラムを組込んで，モニタの助けを借りてデータを収集するものである[3)~6)]．

両方式ともそれぞれ利点と欠点がある．ハードウェア方式はシステムの動作に全く影響を与えないという利点はあるが，同時に収集できるデータの個数や種類に制限があり，また特別な装置を必要とするなど経費の面でも問題がある．一方，ソフトウェア方式は費用の点と，収集データに特に大きな制約がないことは有利であるが，システム動作に何らかの影響を与えることは必至であり，オーバヘッドによるある程度の誤差が生じることも避けられない．しかし，この方式が豊富な情報を多次元的に収集できることは，いくつかの欠点を考慮してもなお大きな利点であると考え，我々はソフトウェア方式を採用することにし，NEAC シリーズ 2200 モデル 700 を対象とした計測システムを開発した[7)]．以下，開発した計測システムの方式設計，実際に得られた測定結果の例および計測システムの応用例について報告する．

2. 計測の対象となるシステムの概要

計測の対象となった NEAC シリーズ 2200 モデル 700（以下モデル 700 という）は，東北大学大型計算機センターに設置されているもので，その主な構成は中央処理装置 (CPU) 1 台* (平均命令実行時間 0.8 マ

† A Design of Software Monitor and Its Application by MASATOSHI MIYAZAKI, SEIZIRO OBATA, and SHIGERU MATSUZAWA (Tohoku University Computer Center).

†† 東北大学大型計算機センター

* 1976 年 8 月までは 2 CPU システム．このため，本計測システムは当初 2 CPU 用として開発したが，現在は 1 CPU 用に一部修正した．

イクロ秒)，主記憶 1 M 字 (128 k 語，1 語 48 ビット，サイクルタイム 0.5 マイクロ秒／語)，ドラム装置 3 台 (4.7 k 字／台，平均アクセス時間 17 ミリ秒，転送速度 387 k 字／秒)，ディスク装置 8 台 (35 M 字／台，平均アクセス時間 60 ミリ秒，転送速度 416 k 字／秒)，その他低速周辺装置となっている．磁気ドラム装置とディスク装置の転送には高速チャネル，その他の周辺装置には標準チャネルが使われる．

ソフトウェア・モニタリングでは割込機能が重要である．モデル 700 には 4 種類の割込機能があり，これらは優先度の高い順に緊急割込み，制御割込み，外部割込み，内部割込みと呼ばれている．緊急割込みはハードウェアの障害時と例外事項の検出時に発生するものであり，システムが正常に稼動しているときには発生しないと考えてよい．制御割込みは主としてタイマ・レジスタから発生するもので，これを時間管理に用いている．外部割込みは周辺装置の信号およびモニタ・コール命令により発生する．内部割込みは記憶保護や不正命令の実行に関連して発生する．

使用しているオペレーティング・システムは東北大学用に開発された MOD 7 と呼ばれるものである．モニタの常駐部は約 140 k 字で主記憶の最下位に置かれる．残りのユーザ領域は固定パーティション方式で使用する．常駐モニタの主なものは割込処理プログラム，時間管理プログラム，入出力制御プログラム，タスク管理プログラムと各種テーブル類で，その他のモニタ・プログラム，たとえばスケジューラ，イニシエータ／ターミネータ，入力ルーチン，出力ルーチン等はユーザ領域で動作するようになっている．

3. 計測システムの概要

ソフトウェア・モニタリングの方式は大きく 2 つに分けられる．1 つはサンプリング方式で[3),5),6)]，これはシステムが動作中の状態を適当な間隔で観測するものであり，主としてシステム資源の使用比率を求めることができる．他はイベント方式と呼ばれるもので[4)]，これは特定の事象の発生ごとにシステムの状態を観測するものである．本計測システムはサンプリング方式に重点を置いているが，簡単なイベント方式も実験的に組み込んだ．

計測システムを作成する方法として 2 通りが考えられる．1 つはモニタにはほとんど手を加えずに，計測プログラムをユーザのプログラムとして動作させる方法である[3),5),6)]．ただし，この場合はモニタから適当な制御を受け渡してもらう必要があり，またユーザのプログラムからモニタ内の必要な情報を自由に参照できることが条件となる．他の方法はモニタを改造して(あるいは開発の当初から)計測に必要な機能を組込んでしまうものである[4)]．この方法はモニタの修正や編集作業に相当の労力を要するが，前者に比べて特に大きな制約もなく，またシステムの信頼性も十分高くできる利点がある．本計測システムは後者の方式をとり，常駐モニタの一部を改造し，さらにモニタの一部として動作する計測プログラムを追加して，モニタの一機能として動作するように設計した．修正したモニタのプログラムは 23 個，新しく追加したプログラムは 22 個である．修正したプログラムの主なものは割込処理，時間管理処理，タスク・スイッチング処理，システム・スタートアップ処理等に関係するものである．追加したものとしては測定システム用のスタートアップ・カード処理，情報の収集処理，収集情報の出力処理等のプログラムである．この修正と追加によりモニタの常駐領域が 4.5 k 字増加した．

この計測システムはシステムのスタートアップ時に計測を行うかどうかの指定ができるようになっており，指定がなければ通常の MOD 7 と同じ動作をするようになっている．このため，修正した各プログラムには計測の有無を判定する命令を入れてある．なお，計測の指定がない場合は，追加したプログラムは主記憶に持込まれないようにしてある．計測指定の際はサンプリング方式かイベント方式のどちらかを選択する必要があり，両方式を同時に動作させることはできない．

計測の開始と終了は任意の時点でコンソール・コマンドで行えるようになっている．したがって，ジョブ群を投入し処理を開始して定常的な状態にしてから計測を始めることができる．また，計測結果は磁気テープに出力するが，センス・スイッチにより出力をバイパスすることができる．つまり，計測は行っていても結果が出力されないわけで，これは長時間の測定で収集データが膨大になる場合，部分的な観測に切換えるのに便利である．

4. サンプリング方式の制御

サンプリング方式において必要な一定間隔の観測時点を得るためにタイマ・レジスタを用いた．このレジスタに値をセットしておくと，2 マイクロ秒ごとに 1 が引かれ値がゼロになると制御割込みが発生する．こ

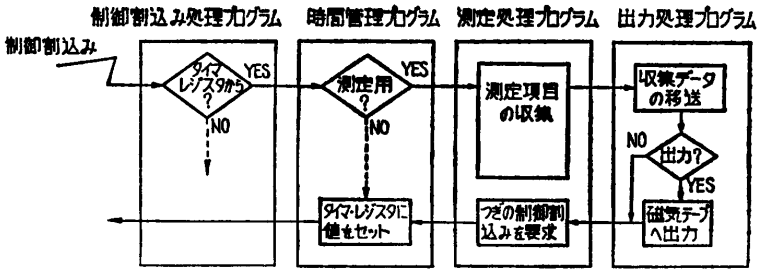

図 1　サンプリング方式の制御
Fig. 1　A control flow of the sampling method.

の割込みは緊急割込みの次に優先度が高いが，緊急割込みは通常の状態では発生しないので，制御割込みが原則として最優先と考えてよい*．したがって他の割込みの影響を受けずに観測時点を設定できる．

サンプリング方式の制御の流れを図1に示す．制御割込みが発生するとそのときの CPU の状態が保存されて，制御割込み処理プログラムに制御が渡される．制御割込みの要因はタイマ・レジスタだけとは限らないので，割込み原因を調べそれがタイマ・レジスタからのものであれば，さらに制御を時間管理プログラムに渡す．このプログラムは MOD 7 のすべての実時間管理を行うものであり，モニタや多数のユーザ・プログラムの実時間の管理要求を1つのタイマ・レジスタを用いて処理している．各プログラムの時間管理要求はタイマ・キューに登録されるようになっており，割込みが発生したときの制御の渡し先も記録されている．

時間管理プログラムは制御割込み処理プログラムから制御を渡されると，それが測定用の割込みであれば制御を測定処理プログラムに渡す．このプログラムではあらかじめ決められた表1のような測定項目をモニタのテーブル類を参照して作業領域に取り込み，制御を出力処理プログラムに渡す．

出力処理プログラムでは測定処理プログラムの作業領域から磁気テープの出力バッファへ収集データを移しブロッキングを行う．1回の収集データは 157 字のレコードになり 52 個のレコードをブロッキングして出力する．磁気テープへの出力は MOD 7 の入出力制御プログラムの制御を受けずに，直接出力命令を使うようにした．これは出力処理のオーバヘッドを減らすためである．収集データのブロッキングあるいは出力バッファの出力を終了すると測定処理プログラムへ制御が戻る．ここで次の観測時点を得るために時間管理要求を出して制御を時間管理プログラムに戻す．

時間管理プログラムではタイマ・キューの中で制御割込みの要求間隔の最小値をタイマ・レジスタにセットする．これは前述のように1つのタイマ・レジスタですべての要求を処理するためである．したがって，次回の観測のための割込み要求がすぐにセットされるとは限らず，中間で別の目的の制御割込みの処理が行われることがあり，測定間隔に多少の誤差が生じることがある．タイマ・レジスタに値をセットしたあと制御割込みプログラムに制御が戻される．このプログラムからの戻り先は制御割込みが発生したときの状態によって異なる．ユーザ・プログラムの処理中であれば制御はディスパッチャに戻される．もしモニタの処理中であれば制御はモニタ内の割込み発生点に戻る．なお，1回の観測における CPU のオーバヘッドは約 500 マイクロ秒であり，さらに 52 回に1回の割合で磁気テープに収集データを掃き出すための処理（バッファの操作）が約3ミリ秒必要である．なお，磁気テープへの出力はエラー・チェックなどはせずおいてきぼり方式にしているので，磁気テープの動作時間の影響はほとんどない．

表 1　サンプリング方式における収集データ
Table 1　The sampling data.

収集データ	目的
(1) プログラム・ステータス・レジスタ	CPU の状態
(2) つぎに実行する命令のアドレス	観測時点がモニタ内かユーザ・プログラム内かの判定
(3) 標準チャネル・ステータス・テーブル	標準チャネルの使用状態
(4) 高速チャネル・ステータス・テーブル	高速チャネルの使用状態
(5) 周辺装置テーブル	ドラム装置，ディスク装置の使用状態
(6) つぎに実行する命令	つぎに実行する命令が参照するデータのアドレスを求める（7章参照）．
(7) インデックス・レジスタの内容	
(8) アドレス・レジスタの内容	
(9) リロケーション・レジスタの内容	

* 実は，モデル 700 には複数の処理モードがあり，各種の割込みとそれが受付けられる優先度の間にやや複雑な関係があるが，ここでは処理モードの説明を省略しても本論文の趣旨に影響はない．

5. イベント方式の制御

イベント方式は特定事象の発生ごとに観測を行うものであるが，今回開発したものは主としてモニタ・プログラムの実行時間を測定することを目的としてお

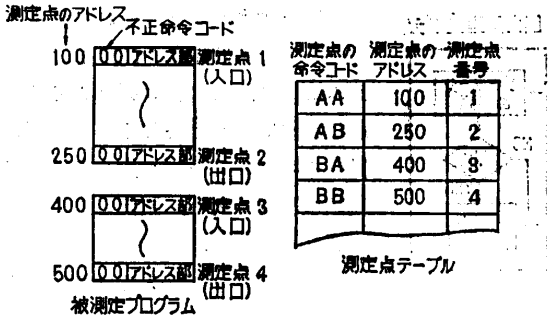

図 2 イベント方式における測定点テーブルの例
Fig. 2 A table of monitoring points.

り，特定事象として不正命令コードの実行を用いた．モニタ・プログラムの中で任意に測定点を設定し，ここに不正命令コードを埋め込んでおく．モニタの処理中に，この命令が実行されると例外事項となり緊急割込みが発生するので，これにより制御を測定処理プログラムに渡す．

プログラムの実行時間を測定するにはその入口と出口を測定点にし，そこでの時刻を調べればよい．測定点に不正命令コードを埋め込む場合，もとの正しい命令コードを保存しておく必要がある．このために図2に示すような測定点テーブルも用いる．この図の例では，プログラム1の入口点のアドレスは100番地であり，ここの命令コードは不正命令コード（00）に置き換えられている．命令のアドレス部はそのままである．もとの正しい命令コード（この例ではAA）は測定点テーブルの命令コードの欄に移され，この命令のアドレスも記録される．テーブルの測定点番号は収集データの解析の際に，測定点の識別のために用いる．

図3はイベント方式の制御の流れを示したものである．測定点の不正命令が実行されると緊急割込みが発生し，制御は緊急割込み処理プログラムに渡る．ここで割込みの原因が不正命令の実行によるものであれば，測定処理プログラムへ制御を渡す．このプログラムではサンプリング方式と同様に測定項目（時刻）を

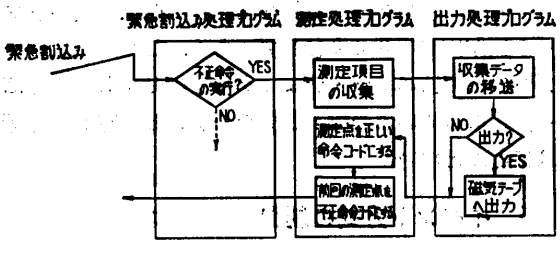

図 3 イベント方式の制御
Fig. 3 A control flow of the event driven method.

取り込み出力処理プログラムへ制御を渡す．出力処理プログラムはサンプリング方式のものと同じものであるが，1回の収集データは63字であり128個のレコードをブロッキングして出力する．なお，磁気テープへの出力はサンプリング方式と全く同じである．

出力処理プログラムから再び測定処理プログラムへ制御が戻ると，ここで緊急割込みの対象となった測定点の不正命令コードを正しい命令コードに置き換える．緊急割込みが発生したアドレスは分かっているので，これと同じアドレスを測定点テーブル上で探すことにより測定点の正しい命令コードを知ることができる．このあと測定点に制御を戻すと，あたかも割込みが発生しなかったのと同じ状態で処理が続行される．つまりこの方法では正しい命令コードの実行は測定点で行われるので，次にその命令が実行される前に，その命令コードを再度不正命令コードに置き換えておかなければならない．このため測定処理プログラムでは前回の処理で正しい命令コードに置き換えたところを再び不正命令コードに置き換える操作も行う．このあと制御を緊急割込み処理プログラムに戻して測定点の実行に移る．

測定点の指定はシステムのスタートアップ時に行い，不正命令コードの埋め込みや測定点テーブルの作成はすべてこの時点で行っておく．測定点は最大30個まで指定できる．被測定プログラムが入口点や出口点を複数個もつ場合があるので，測定点の設定には注意が必要である．また，入口点と出口点の間で別のプログラムを呼び出して実行が移る場合は，呼び出されるプログラムにも測定点を設けておき，解析の際にこのプログラムの実行時間を補正すればよい．

6. 計 測 例

ここでは本計測システムを実際に動作させたときの測定結果についていくつかの例を示す．測定には通常の運用で処理されるジョブのパターンに近いテスト・ジョブ群を用いた．図4と表2～4はサンプリング方式の場合の結果である．サンプリング間隔はいずれも100ミリ秒とした．表5はイベント方式による結果である．

表2とこれを図示した図4は①～⑧の8つの項目を用いてシステムの動作状況を表現したものである*．測定方法はつぎの通りである．全部のテスト・ジョブをあらかじめ入力ファイルに投入しておく．ジョブの結果は出力ファイルへ出力する．入力ファイルと出力

* KIVIAT グラフと呼ばれている．

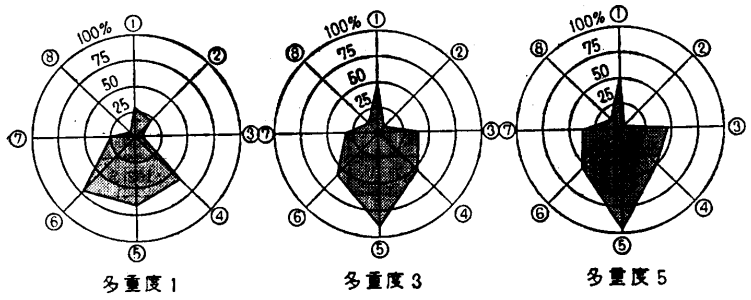

図 4　リソース使用率の図式表現
Fig. 4　A graphical description of resource usage.

表 2　CPU と高速チャネルの使用率
Table 2　A percentile usage of CPU and high speed channel.

	多重度 1	多重度 3	多重度 5
① CPU Active	28.6%	43.2%	48.0%
② CPU Only	22.0	5.4	1.8
③ CPU/Channel Overlap	6.6	37.8	46.2
④ Channel Only	59.2	52.6	49.2
⑤ Channel Busy	65.8	90.4	95.4
⑥ CPU Wait	71.4	56.8	52.0
⑦ User State	22.1	32.0	36.1
⑧ Monitor State	6.5	11.2	11.9

ファイルはディスク装置上に確保されている．測定中はカード読取り機とラインプリンタは動作させない．したがって，処理中の入出力要求はすべてディスク装置が接続されている高速チャネル（1個）に対するものだけとなる．ドラム装置は使用しない．多重度を 1, 3, 5 と変えて測定した．

項目の値は CPU とチャネルの状態を測定時間に対する比率（%）で表わしたものである．CPU の状態は①の動作と⑥の遊びに分けられ，①はさらに⑦のユーザ・プログラムの処理と⑧のモニタの処理に分けられる．チャネルの状態としては⑤の動作があり，②～④は CPU とチャネルが同時動作しているかどうかを示す．項目の値は奇数番が大きく偶数番が小さいほど，システムの効率としては望ましいことになる．つまり，図 4 においては円中央の黒い部分が十字形に近くなることが望ましいわけである．多重度が 1 の場合に比べて多重度 3 では形がいくらか改善されているが．3 と 5 では大きな差異はない．多重度を大きくしても④と⑥の項目の値があまり小さくならないのは，CPU とチャネルの処理能力がアンバランスであることを示している．

表 3 は多重度 3 でジョブ処理を行い，同時にカード読取り機（入力ファイルへのジョブの入力）とラインプリンタ（出力ファイルからの結果の出力）をそれぞれ 1 台動作させたときの測定例である．高速チャネルにはジョブから出される入出力要求のほかにジョブの入力および結果の出力に関連する入出力要求も加わることになる．CPU の動作はモニタ，入力プログラム，出力プログラムおよびユーザ・プログラム（パーティション 1～3）に分けられる．入力と出力プログラムに使われる CPU は比較的少ない．また，3 つのパーティションに CPU が配分されている様子がよく分かる．カード読取り機とラインプリンタは別々の標準チャネルを使っており，このチャネルの使用率からこれらの装置の動作状態が分かる．ラインプリンタは相当よく稼動している．高速チャネル（ディスク装置用）の使用率は 100% に近く，これがネックになっていることは明らかである．

表 4 は表 3 の測定における入出力要求のディスク装置の各ユニットに対する割合を示したものである．ユニット 1 にはシステム・プログラムが置かれているので入出力要求の割合が大きくなっている．他のユニットには入力ファイル，出力ファイル，ログ・ファイル，ワーク・ファイル，ライブラリなどが置かれており，使用の比率に大きな差はない．

次にイベント方式の測定を示す．この方式でモニ

表 3　プログラムの CPU 使用率と周辺装置のチャネル使用率
Table 3　CPU usage for programs and channel busy rate on peripheral units.

CPU Active	モニタ・プログラム	14.92%
	入力プログラム	0.19
	出力プログラム	0.21
	パーティション 1	4.29
	パーティション 2	6.17
	パーティション 3	3.67
Channel Busy	カード読取り機	24.66
	ラインプリンタ	75.10
	ディスク装置	98.24

表 4　ディスク装置の使用率
Table 4　Usage ratio of disk units.

ディスク装置	1	2	3	4	5	6	7	8	計
使用率(%)	51.8	6.1	8.1	9.8	4.7	8.2	6.4	4.9	100.0

表 5 モニタ・プログラムの実行時間と平均使用回数の例

Table 5 An example of execution times and usage frequency of monitor programs.

モニタ・プログラム	機能	平均実行時間 (μs)	平均使用回数 (回/秒)		
			多重度 1	多重度 3	多重度 5
ディスパッチャ	タスクへの CPU の割当て	52	102.7	145.7	162.2
シリアル制御	資源のシリアル制御	88	0.8	1.0	1.3
タスクの同期処理 1	事象待ちタスクの制御	71	23.7	31.3	35.4
タスクの同期処理 2	事象完了後のタクスの制御	120	29.2	36.9	40.5
標準出力制御	システムの出力ファイルの制御	66	17.3	29.7	32.4

タ・プログラムの時間を測定するには，最初に測定のオーバヘッド（1つの測定点の処理に要する時間）を求める必要がある．これには2つの測定点を連続したアドレスに置いて測定を行えばよい．こうすることにより被測定プログラムの実行時間は零になるので，2つの測定点の時間差がオーバヘッドとなる．この方法で求めたオーバヘッドの平均は 250 マイクロ秒であった．**表5**はモニタ・プログラムの中の代表的な5つのプログラムの平均実行時間と処理中の平均使用回数を示したものである．平均実行時間は多重度1で測定したものであり，この値は多重度を変えても変化しない．使用回数は多重度が増加するにつれて多くなるが，これは当然のことである．

7. プログラム波形の観測への応用

サンプリング方式でシステムの効率を測定するのは，システム資源の使用率を求めるのが主目的であるから，一般にサンプリング間隔は比較的大きな値を用いる．もしこの間隔を極めて小さな値に設定して観測を行えば，システム動作の中で CPU の動きを時々刻刻把握できるはずである．

CPU がプログラムを実行する際，実行される命令の置かれているアドレス（アドレス空間）は時間の経過とともに順次変化していく．したがって，アドレス空間の変化の様子を微小時間でとらえれば，プログラム動作は時間とプログラムのアドレス空間の関数として表現され，これを一種の波形（以下プログラム波形という）として観測することが可能となる．この章では測定システムを応用したプログラム波形の観測について述べる．

タイマ・レジスタにセットできる値の最小値は2マイクロ秒であるが，時間管理プログラムで値をセットしてから制御割込み処理プログラムに戻り，さらにこのプログラムから制御が離れるまでに約 70 マイクロ秒の時間が必要である．観測の対象となるプログラムはユーザのプログラムとして動作する必要があり，このため制御割込み処理プログラムからの制御は一旦ディスパッチャなどのモニタ・プログラムに戻され，その後被観測プログラムの実行に CPU が割当てられる．この間に費やす時間は 60～80 マイクロ秒程度である．したがって，毎回の割込みが必ず被観測プログラムの実行中に起きるようにするには，タイマ・レジスタにセットする割込み発生間隔を 150 マイクロ秒以上にしなければならない．この間隔から前述の時間を差し引いた時間が実際のサンプリング間隔（以下，実効サンプリング間隔という）となる．以上の説明を図示したのが**図5**である．

このように，タイマ・レジスタのセットから被観測プログラムの実行に移るまでの時間は変動し，また割込みの検出時点はそのとき実行していた命令の終了時であるから，実効サンプリング間隔にはゆらぎが生じるのは当然である．したがって，これは平均値として求めることになる．割込みの発生間隔を 150 マイクロ秒に設定すると，平均実効サンプリング間隔は 4.9 マイクロ秒となる．これは一連の同一命令を繰り返し実行するプログラムを用いて測定した．モデル 700 の平

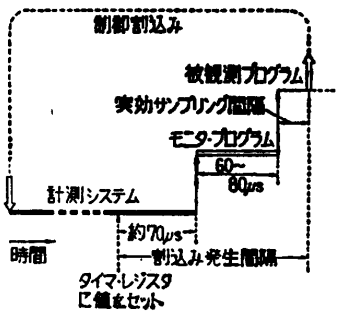

図 5 プログラム波形の観測における時間の関係
Fig. 5 A time sequence of program wave observation.

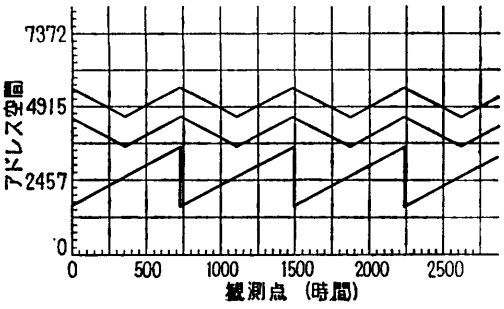

図 6 典型的なプログラム波形
Fig. 6 A typical program wave.

均命令実行時間は 0.8 マイクロ秒であるので，このサンプリング間隔 4.9 マイクロ秒は少し大き過ぎるが，プログラムの実行中に極端にアドレス空間が変化しないと仮定し，ある程度の誤差を認めるなら近似的なプログラム波形を観測することは可能と思われる．

各観測時点において，つぎに実行される命令が置かれているアドレスをその時刻における命令のアドレス空間とみなす．さらに，この命令が実行時に参照するデータ領域のアドレスも考慮する必要があるが，これは表 1 の収集データ(6)～(9)を合成することによって求めることができる．ただし，間接アドレスが使われている場合は，途中の参照アドレスを求めることはできない．なお，データを参照しない命令もあるので，このときの波形の取扱いは注意が必要である．

図 6 は同一命令で構成される命令群を単純に繰り返し実行するようなプログラムの波形の例であり，磁気テープに記録された観測データを作図装置に出力したものである．横軸の数字はサンプリング点の番号を示し，縦軸の数字はアドレス（単位は字，1 字＝6 ビット）を示す．一番下の波形は命令のアドレス空間を示しており，これは明らかにのこぎり波である．サンプリング点の間隔は 4.9 マイクロ秒であるから，周期は約 3.7 ミリ秒となる．一般に命令のプログラム波形はこのようなのこぎり波で合成されることは容易に想像できる．上の 2 つの波形は命令が参照するデータのアドレス空間であり，強調のために三角波となるようにしたものである．命令は 2 アドレス方式のものを使っているので 2 つのアドレス空間が記録される．この図の例から，プログラムの動作を波形として観測することは十分可能であり，しかもその波形がプログラムの論理的な動作とよく対応することが分かる．

図 7 は FORTRAN コンパイラのプログラム波形の一部分である．実線は命令のアドレス空間を示し，○印と×印は参照データのアドレス空間を表わしている．この例では命令のアドレス空間はほぼ規則的に変化しており，何か特定の繰返し処理を行っていることが分かる．また，プログラムの命令部分（上方）とデータ部分（下方）ははっきり分けられており，データ部分を参照しながら処理が行われている様子がよく現われている．

8. むすび

本稿では，ソフトウェアの手法を用いたシステム評価のため計測について，その方式設計を述べるとともに，いくつかの測定結果を示した．さらに，この計測

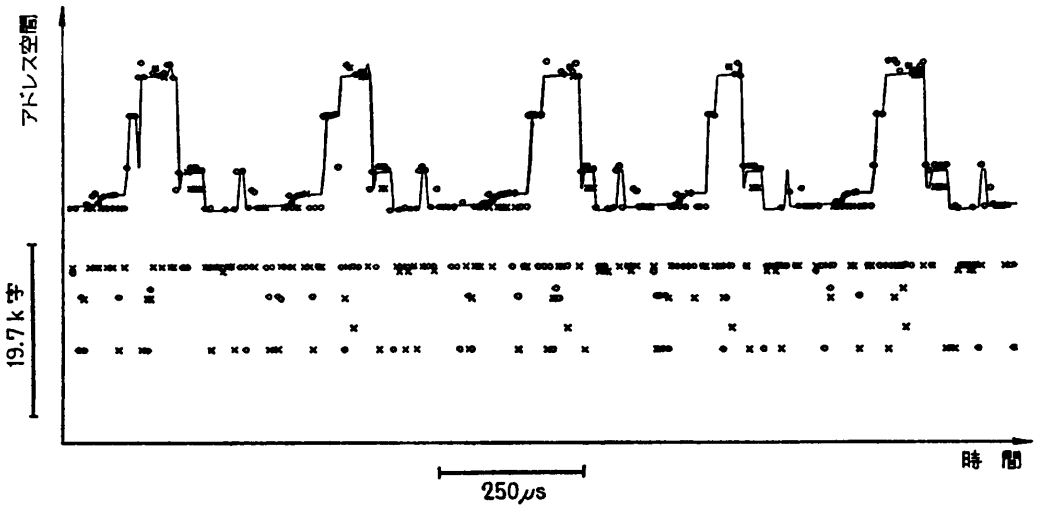

図 7 プログラム波形の例 (FORTRAN コンパイラ)
Fig. 7 An example of program wave (FORTRAN Compiler).

システムを応用したプログラム波形の観測についても述べた.

計測システムをモニタの内部に組込む方式を採用したので，その開発と試験には相当の時間と労力を要したが，収集できる情報に特に制約はなく，また計測システムの修正等も容易に行えるので，それなりの効果はあったと思われる．実際，サンプリング方式の測定項目のうち(6)～(9)に関しては，プログラム波形の観測のために後から追加したものである．

サンプリング方式の制御に関しては特に問題となる部分はないと思われるが，1つしかないタイマ・レジスタを共用しているので，実際の測定間隔には変動が生じる．このほか，1回の観測における CPU のオーバヘッドにより，サンプリング方式には本質的にある程度の誤差が生じるのは避けられない．しかし，この方式はシステム資源の使用比率を求めるのが目的であるから，測定時間とサンプリング間隔を適当に選べば，十分信頼できるデータが得られるものと思われる．

イベント方式は実験的に組み込んだものであり，その機能もモニタ・プログラムの実行時間を測定することに限定した．したがって，イベントとして不正命令の実行のみを用いているが，一般には他の種々のイベントも用いた総合的なイベント方式が望ましい．

プログラム波形の観測は，この計測システムの設計当初は予想していなかった応用である．プログラムの動作を近似的に波形として観測できるということは，プログラム動作の解析に波形解析の一般的手法が応用できることになる．たとえば自己相関関数やパワースペクトルを求めることによって，プログラムの特徴を定量的にとらえることができれば，プログラム波形はプログラムの評価に関する研究の有力な手法になるものと思われるが，これについては今後の研究課題としたい．

謝辞 本計測システムは日本電気(株)の菊池豊彦，青山博牟両氏との共同研究により設計開発されたものである．両氏に厚くお礼申し上げる．また，システムの開発・試験に協力いただいた日本電気(株)の横山幸男氏に深謝する．

参考文献

1) R. J. Rund: The CPM-X—A Systems Approach to Performance Measurement, Proc. FJCC, pp. 949-957 (1972).
2) 箱崎，小野：ハードウェア・モニタによるシステム測定，情報処理，Vol. 13, No. 11, pp. 782-788 (1972).
3) H. N. Cantrell and A. L. Ellison: Multiprogramming System Performance Measurement and Analysis, Proc. SJCC, pp. 213-221 (1968).
4) H. D. Schwetman and J. C. Brown: An Experimental Study of Computer System Performance, Proc. ACM, pp. 693-703 (1972).
5) J. M. Schwartz and D. S. Wyner: Use of the SPASM Software Monitor to Evaluate the Performance of the Burroughs B 6700, Proc. NCC, pp. 93-100 (1973).
6) 北川，萩原，金沢，藤原：オペレーティング・システムのパフォーマンス・モニタリング，情報処理，Vol. 13, No. 11, pp. 789-797 (1972).
7) 松沢，小畑，宮崎，青山，菊池：ソフトウェアによるシステム評価のための計測について，情報処理学会第14回大会予稿集 (1973).

(昭和52年3月3日受付)
(昭和53年5月29日採録)