stichting
mathematisch
centrum

MC

AFDELING INFORMATICA IN 16/79 MEI

P. KLINT & T. HAGEN

CRITERIA VOOR EEN RESEARCHCOMPUTER

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

Criteria voor een researchcomputer

door

P. Klint & T. Hagen

ABSTRACT

In dit rapport worden de eisen voor een speciale researchcomputer voor de afdeling Informatica van het Mathematisch Centrum samengevat.
Tevens wordt verslag gedaan van een vergelijkend onderzoek van PDP11/45, VAX11/780 en Interdata 8/32.

Eerste druk Februari 1979
Tweede druk Mei 1979

INHOUD

1. Doel en eisen 1
1.1. Eisen voortvloeiend uit het meerjarenplan 1
1.1.1. Programmaoverdracht 2
1.1.2. Tussentalen 2
1.1.3. Nieuwe programmeertalen en datastructuren 3
1.1.4. Picture processing 4
1.2. Technische eisen 4
1.2.1. Reken- en geheugensnelheid 4
1.2.2. Geheugenorganisatie 4
1.2.3. Achtergrondgeheugen 5
1.2.4. Woordlengte en adresseerbaarheid 5
1.2.5. Hardware compatibiliteit 5
1.2.6. Hogere programmeertalen en instructie repertoire 5
1.2.7. Microprogrammeerbaarheid 5
1.2.8. Bedrijfssysteem 6
1.2.9. Uitbreidbaarheid 6

2. Vergelijkingsmethode 6
2.1. Procedure en voorwaarden 6
2.2. Benchmarks 7
2.2.1. B1: Byte-manipulatie 8
2.2.2. B2: Bit-manipulatie 8
2.2.3. B3: Procedure-aanroepmechanisme 8
2.2.4. B4: Floatingpoint operaties 8

3. Algemene gegevens en resultaten 8
3.1. PDP11/45 8
3.1.1. Hardware 9
3.1.2. Benchmark resultaten 9
3.2. PDP11/60 10
3.2.1. Hardware 10
3.2.2. UNIX 10
3.2.3. Conclusie 10
3.3. VAX11/780 10
3.3.1. Hardware 10
3.3.2. UNIX 11
3.3.3. Benchmark resultaten 11
3.3.4. Conclusie 12
3.4. Interdata 8/32 13
3.4.1. Hardware 13
3.4.2. UNIX 14
3.4.3. Benchmark resultaten 14
3.4.4. Conclusie 14
3.5. Samenvatting 15

Referenties 16

Appendix I Benchmark 1 C-programma 17
Appendix II Benchmark 2 C-programma 19
Appendix III Benchmark 3 C-programma 22
Appendix IV Benchmark 4 C-programma 23
Appendix V Procedure ingang- en uitgangmechanisme 25
Appendix VI Benchmark 1 PDP11 assembler programma 26

Appendix VII Benchmark 2 PDP11 assembler programma 28
Appendix VIII Benchmark 3 PDP11 assembler programma 31
Appendix IX Benchmark 4 PDP11 assembler programma 33
Appendix X Benchmark 1 VAX11 assembler programma 35
Appendix XI Benchmark 1 VAX11 hercodering 37
Appendix XII Benchmark 2 VAX11 assembler programma 39
Appendix XIII Benchmark 2 VAX11 hercodering 41
Appendix XIV Benchmark 3 VAX11 assembler programma 43
Appendix XV Benchmark 4 VAX11 assembler programma 45
Appendix XVI Benchmark 1 Interdata 8/32 assembler programma 47
Appendix XVII Benchmark 1 Interdata 8/32 hercodering 51

1. Doel en eisen.

Binnen het Mathematisch Centrum is de behoefte ontstaan om naast de PDP11/45, die voor het Computer Graphics project gebruikt wordt, de beschikking te krijgen over een speciale computer voor researchdoeleinden. Deze computer is bedoeld voor onderzoek, dat niet op andere wijze (b.v. via een universitair rekencentrum) gerealiseerd kan worden. In eerste instantie worden de eisen, die aan deze computer gesteld worden, bepaald door de bestaande onderzoeksprojecten, zoals geformuleerd in het meerjarenplan van het Mathematisch Centrum [2].

Dit eisenpakket beperkte het aantal computers waaruit gekozen kon worden. Bovendien heeft de afdeling Informatica van de Katholieke Universiteit te Nijmegen reeds een vergelijkend onderzoek [1] gedaan, waarvan wij gebruikt hebben gemaakt. Nadat op deze wie het aantal potentiële candidaten gereduceerd was, bleven de volgende serieuze candidaten over:

- B1800 (Burroughs)
- PDP11/60 (Digital Equipment Corp., DEC)
- VAX11/780 (Digital Equipment Corp., DEC)
- Interdata 8/32 (Perkin-Elmer)

Om redenen van kostprijs (B1800) en van mogelijke geheugengrootte (PDP11/60) werd het onderzoek in een later stadium beperkt tot de VAX11/780 en de Interdata 8/32. Toen de keuze eenmaal tot deze twee computers beperkt was, is de uiteindelijke keuze bepaald door middel van een aantal benchmark programma's en bestudering van de implementatie van het bedrijfssysteem UNIX op beide machines.

1.1. Eisen voortvloeiend uit het meerjarenplan.

De onderzoeksprojecten of onderdelen daarvan, die relevant zijn voor de apparatuurkeuze, kunnen globaal ingedeeld worden in:

- Onderzoek van programmaoverdracht en machineonafhankelijkheid.
- Ontwerp en implementatie van tussentalen.
- Ontwerp en implementatie van nieuwe programmeertalen.
- Onderzoek van nieuwe datastructuren.
- Stringmanipulatie en patroonherkenning.
- Picture processing.

Deze onderwerpen zullen nu nader besproken worden. Hierbij wordt steeds een specifiek deelproject gekarakteriseerd en worden vervolgens de eisen en wensen opgesteld die, op grond van de typering van het deelproject, van belang zijn bij de apparatuurkeuze.

1.1.1. Programmaoverdracht.

Voor het onderzoeken van programmaoverdracht is het noodzakelijk dat:

- Deze overdracht naar zeer uiteenlopende computers daadwerkelijk uitgevoerd wordt.

- De specificaties van verschillende computers onderzocht worden.

Aan beide eisen kan voldaan worden door een groot aantal (mini- of micro-) computers van uiteenlopend fabrikaat aan te schaffen, dan wel een flexibele microprogrammeerbare computer te kiezen, die emulatie van zeer verschillende computers toestaat.

1.1.2. Tussentalen.

Bij een aantal projecten wordt gebruik gemaakt van zogenaamde tussentalen, te weten:

- Alice, de tussentaal die gegenereerd wordt door de machineonafhankelijke ALEPH compiler.

- De machineonafhankelijke objectcode van de ALGOL 68 compiler.

- ILP, Intermediate Language for Pictures, een tussentaal die in het Computer Graphics project gebruikt wordt voor de machineonafhankelijke beschrijving van tekeningen.

De eerste twee tussentalen worden gebruikt voor programmaoverdracht en richten zich op meer dan één machinearchitectuur. Dit komt bijvoorbeeld tot uiting in een grote mate van redundantie. Deze categorie tussentalen legt eisen op als beschreven in de voorgaande sectie.

Voor tussentalen is om redenen van efficiëntie nog een vertaalstap nodig, voordat programma's in deze talen tenuitvoer gebracht kunnen worden. Deze vertaalstap bemoeilijkt in feite ontwerp en implementatie van (onder andere) deze tussentalen. In het geval van ILP is er bovendien sprake van interactie, doordat tekeningen en dus hun ILP-beschrijving interactief gewijzigd kunnen worden. Een dergelijke interactie wordt aanzienlijk vereenvoudigd als ILP programma's zonder compilatiestap direct tenuitvoer gebracht kunnen worden.

Tussentalen corresponderen met een abstracte machine, die ideaal is voor het executeren van die tussentaal en dus voor de bijbehorende hogere taal. Tussentalen hangen nauw samen met machinearchitectuur. Anderzijds hebben hogere programmeertalen invloed op de vorm van tussentalen. Het onderlinge verband tussen

hogere programmeertalen, tussentalen en machinetalen wordt eveneens onderzocht.

1.1.3. Nieuwe programmeertalen en datastructuren.

Naast ontwerp en implementatie van bovenstaande (lager niveau) talen wordt ook gewerkt aan het ontwerp van hoger niveau talen:

- De taal B, een eenvoudige, gestructureerde, conversationele programmeertaal.
- De taal ABSTRACTO, voor de formulering van abstracties die aan algoritmen ten grondslag liggen.
- ALGOL 68G, een grafische uitbreiding van ALGOL 68.
- SPRING, een taal voor stringmanipulatie en patroonherkenning.
- Talen voor picture array processors (zie 1.1.4.).

Tot op dit moment hebben technologische beperkingen grote invloed gehad op controle- en datastructuren van programmeertalen, zoals b.v.:

- Eindige geheugen- en integercapaciteit.
- Sterke nadruk op het sequentieel doorlopen van programma's, waardoor in veel gevallen overspecifikatie van de volgorde ontstaat.
- Het (om redenen van efficiëntie) vermijden van associative zoekoperaties.

Het is van groot belang dat in de toekomst de invloed van dergelijke beperkingen op taalontwerp en programmeermethoden verminderd wordt. Om deze redenen is het gewenst een computer te kiezen, die op een of meer punten de traditionele hardwarebeperkingen overschrijdt (b.v. associatief geheugen, microprogrammeerbaarheid, parallelle processors), dan wel gemakkelijk aangepast kan worden aan specifieke probleemstellingen. Voor het onderzoeken van datastructuren is het gewenst te beschikken over een geheugen dat op willekeurige manier kan worden ingedeeld (d.w.z. hardware bit-adresseerbaarheid).

Bovendien is het gewenst dat er hardware faciliteiten zijn voor het verrichten van metingen aan programmatuur, zodat een beter inzicht verkregen kan worden in de gedragingen daarvan. Een dergelijke externe hardware monitor maakt het mogelijk om programma's (of gedeelten daarvan) te onderzoeken zonder het runtime gedrag daarvan te beïnvloeden of hercompilatie te vereisen.

1.1.4. Picture processing.

Een belangrijke manier van grafische invoer is het omzetten van een visueel beeld in een z.g. beeldmatrix van lichtwaarden. Deze beeldmatrix kan met behulp van een aantal (parallelle) patroonherkenningsprocessen omgevormd worden tot informatie die door een grafisch systeem verwerkt kan worden. Een tweedimensionaal netwerk van parallelle processors die een beeldmatrix analyseren, wordt aangeduid als Picture Array Processor (PAP). In dit verband worden locale, parallelle algoritmen onderzocht, die mogelijk in een PAP besturingstaal ingepast zullen worden.

De ontwikkeling van algoritmen en efficiëntiecriteria is sterk afhankelijk van de bruikbaarheid en beschikbaarheid van bestaande (micro- en array-) processoren. Een flexibele researchcomputer kan hier in hoge mate aan bijdragen.

1.2. Technische eisen.

Op grond van de eisen voortvloeiend uit het meerjarenplan kunnen een aantal globale eisen geformuleerd worden waaraan de nieuwe apparatuur dient te voldoen. Bovendien zijn er eisen die voortvloeien uit compatibiliteitsoverwegingen.

1.2.1. Reken- en geheugensnelheid.

Reken- en geheugensnelheid moeten vergelijkbaar zijn met die van de huidige PDP11/45. Men moet hierbij denken aan een gemiddelde instructietijd van 0.5-1.0 microseconde en een effectieve geheugenaccestijd van circa 300 nanoseconde. De meer rekenintensieve (grafische) programma's zullen op de PDP11/45 blijven draaien of (voor zover mogelijk) via een binnenkort te realiseren communicatiekanaal met het academisch rekencentrum (SARA) op de CYBER 173 tenuitvoer gebracht worden. Snelle floatingpoint operaties zijn daarom niet noodzakelijk, al is een hardware floatingpoint eenheid toch gewenst.

1.2.2. Geheugenorganisatie.

In een omgeving waar software ontwikkeld wordt is het gewenst dat programma's van (vrijwel) willekeurige grootte gemaakt kunnen worden, zonder dat daarvoor speciale maatregelen (b.v. overlays) nodig zijn. De (2*) 64 Kbytes adresruimte van de PDP11/45 is soms al te klein. Om deze reden is het gewenst dat de hardware faciliteiten biedt voor het realiseren van een virtueel-geheugensysteem. Het is duidelijk dat systeemprestaties snel afnemen naarmate er meerdere grote programma's gelijktijdig actief zijn. De ervaring heeft geleerd dat minstens 128 Kb (fysisch) geheugen nodig is. Om nog enige speelruimte te hebben is 256 Kbytes gewenst. Zo mogelijk moet de hoeveelheid geheugen zelfs dan nog uitgebreid kunnen worden.

1.2.3. Achtergrondgeheugen.

Als men uitgaat van 15 gebruikers en 2 Mbytes opslagruimte per gebruiker is 30 Mbytes achtergrondgeheugen nodig. Voeg hierbij ruimte nodig voor swapping en de mogelijkheid voor enige groei dan is circa 60 Mbytes vereist. Een klein gedeelte hiervan moet snel toegankelijk zijn (circa 5 Mbytes met gemiddelde accestijd kleiner dan 10 milliseconde en informatietransportsnelheid kleiner dan 2.5 microseconde/byte)

1.2.4. Woordlengte en adresseerbaarheid.

De woordlengte moet minimaal 16 bits zijn (bestaande programma's vereisen dit), maar 32 bits is meer wenselijk. Byte-adressering binnen een woord is noodzakelijk om efficiënte karaktermanipulatie mogelijk te maken. Mogelijkheden voor protectie en sharing van geheugensegmenten is vereist.

1.2.5. Hardware compatibiliteit.

De reeds aanwezige randapparatuur moet door de nieuwe computer via UNIBUS interfaces (eventueel met adaptor) bestuurd kunnen worden. Dit heeft tevens het voordeel dat voor de nieuwe installatie gekozen kan worden uit de grote markt van PDP11-compatibele randapparatuur.

1.2.6. Hogere programmeertalen en instructie repertoire.

Op de nieuwe installatie zal nauwelijks in assembler geprogrammeerd worden. Het is derhalve wenselijk dat de architectuur van deze computer efficiënte verwerking van in hogere programmeertalen geschreven programma's toelaat. Speciale aandacht verdienen hierbij: het procedure call/return mechanisme, array-indicering, loop-instructies en de mogelijkheden voor het manipuleren met datastructuren. Het is zeer belangrijk dat het instructierepertoire "compleet" is, in de zin dat alle operaties voor conversie van/naar de standaard hardware datatypes aanwezig zijn en bovendien iedere operatie (add, multiply, shift enz.) op alle hardware datatypes toegepast kan worden. Een dergelijk instructierepertoire vereenvoudigt het schrijven van codegenerators aanzienlijk.

1.2.7. Microprogrammeerbaarheid.

Ten behoeve van het project AI6 (computerarchitectuur en taalontwerp [2]) en voor verfijnde afstemming van de installatie op specifieke applicaties is een zekere mate van microprogrammeerbaarheid gewenst. Het is niet nodig dat complete nieuwe architecturen geëmuleerd kunnen worden; voor dit doel kunnen in de toekomst goedkope microprogrammeerbare microcomputers aangeschaft worden. Het is echter wel wenselijk dat nieuwe instructies kunnen worden toegevoegd (door middel van microprogrammering) om hetzij een specifiek aspect van een nieuwe architectuur te onderzoeken, hetzij applicatiegerichte instructies toe te voegen. Het is belangrijk dat de microprogrammerings-

faciliteiten voor gebruikersprogramma's toegankelijk zijn, mits gebruikersprogramma's onderling beschermd zijn tegen eventuele fouten in de microcode.

1.2.8. Bedrijfssysteem.

Op de huidige installatie wordt, tot onze grote tevredenheid, het timesharing systeem UNIX [3] gebruikt. De opzet van dit systeem is zodanig, dat het gemakkelijk aangepast kan worden aan de plaatselijke eisen. Programmatuur voor het aansluiten van randapparatuur, maar ook speciale systeemprogrammatuur kan met relatief weinig mankracht worden ingepast. Op het Mathematisch Centrum is een gedegen kennis van UNIX aanwezig. Op de nieuwe installatie dient dan ook zo mogelijk hetzelfde systeem beschikbaar te zijn. Deze eis is voor ons bijzonder zwaarwegend, temeer daar onze beperkte mankracht uitgebreide conversie van bestaande programma's niet toelaat.

1.2.9. Uitbreidbaarheid.

Fysisch geheugen, writable control store (microprogrammageheugen) en achtergrondgeheugen moeten uitgebreid kunnen worden.

2. Vergelijkingsmethode.

Om een gedegen inzicht te krijgen in de eigenschappen van de aangeboden computers werden een aantal programma's (benchmarks) geschreven. Aan de leveranciers is gevraagd deze programma's op hun computersysteem te implementeren.

2.1. Procedure en voorwaarden.

Aan de door de leverenciers te implementeren programma's zijn de volgende eisen gesteld:

- De programma's dienen met behulp van dezelfde algoritmen, dezelfde problemen op te lossen als de door het Mathematisch Centrum verstrekte programma's. M.a.w. de programma's moeten functioneel hetzelfde zijn. Er is geen interesse in de snelste methode om de voorgelegde problemen op te lossen.

- Het procedure-aanroepmechanisme mag afwijken van de voorgestelde manier, onder voorwaarde dat:

  - Recursieve procedure-aanroepen toegestaan zijn.

  - In iedere procedure tenminste drie registers vrij beschikbaar zijn. Deze registers moeten automatisch worden bewaard bij procedure-ingang en worden hersteld bij procedure-uitgang.

- De programma's dienen de standaardinstructies van de betreffende machine te gebruiken en mogen geen gebruikmaken van additionele gemicroprogrammeerde instructies.

De leveranciers is gevraagd om de tekst van de door hun geïmplementeerde programma's te verschaffen, evenals executietijden en gedetailleerde informatie over de gebruikte configuraties. De executietijden dienden zo mogelijk gespecificeerd worden als volgt:

- Totale executietijd: de tijd verstreken tussen het geven van het commando om een programma te executeren en de systeemindicatie, dat de executie voltooid is.

- Systeemoverhead: de tijd die het bedrijfssysteem nodig heeft om executie van een programma mogelijk te maken. Dit betreft tijd voor het laden, afhandelen van de interrupts, swappen en timing.

- Reële executietijd: de totale tijd die de CPU ten behoeve van een programma actief is geweest.

Verder is gevraagd een indicatie te geven van de grootte (in bytes) van de verschillende programma's. Bovendien is verzocht om, indien er voor de executie van de programma's gebruikt wordt gemaakt van procedures uit programmatheken, eveneens de grootte van dergelijke procedures aan te geven.

De volgende informatie werd gevraagd over de configuratie, waarop de benchmarks tenuitvoer gebracht zijn:

- type van de CPU

- grootte en snelheid van het geheugen.

- grootte en snelheid van het cachegeheugen (indien aanwezig).

- type van de floatingpoint hardware.

- type van het swapping device (indien aanwezig).

- het gebruikte bedrijfssysteem, waar alle metingen onder gedaan zijn.

2.2. Benchmarks.

Uit het eisenpakket (zie 1.2.) kan geconcludeerd worden welke eigenschappen belangrijk zijn. Het overgrote deel van het rekenwerk zal bestaan uit byte-manipulatie. Andere belangrijke rekeneigenschappen zijn de snelheid van het procedure-aanroepmechanisme

bit-manipulatie en floatingpoint operaties. Een viertal benchmark programma's, die deze eigenschappen beproeven, is geschreven in de hogere programmeertaal "C" [4]. Alle vier benchmark programma's zijn vertaald naar de assembleertaal van de PDP11 en zodaning van commentaar voorzien dat kennis van PDP11 instructies niet vereist is om de programma's te begrijpen.

2.2.1. B1: Byte-manipulatie.

Probleem 1 bestaat uit het roteren van een array van 500 karakters. Na 500 rotaties moet het array (uiteraard) uit dezelfde karakters bestaan als voor aanvang van de rotaties. Het bovenstaande proces moet 40 maal uitgevoerd worden.

2.2.2. B2: Bit-manipulatie.

Probleem 2 bestaat uit het roteren van een array van 500 bits. Na 500 rotaties moet het array (uiteraard) uit dezelfde rij bits bestaan als voor aanvang van de rotaties. Het bovenstaande moet tweemaal uitgevoerd worden. De nadruk bij dit probleem ligt op bit-adressering en bit-operaties.

2.2.3. B3: Procedure-aanroepmechanisme.

Probleem 3 bestaat uit het berekenen van de sterk recursieve functie van Ackermann voor verschillende argumentwaarden.

2.2.4. B4: Floatingpoint operaties.

Probleem 4 bestaat uit het 360 maal roteren van de vector (1.0, 0.0) over een hoek van 1 graad. Hierna mag de resulterende vector maar binnen een bepaalde tolerantie (0.00001) verschillen van de oorspronkelijke vector.

3. Algemene gegevens en resultaten.

De programmateksten van de benchmarks zijn opgenomen in de appendix. Voor zover deze van de leveranciers afkomstig zijn, kunnen er fouten in geslopen zijn bij het reproduceren. De vertegenwoordiger van Interdata (Perkin-Elmer) heeft alleen de tekst van de eerste benchmark aan ons verstrekt.

3.1. PDP11/45.

Als referentie is de PDP11/45 in dit onderzoek opgenomen.

3.1.1. Hardware.

De algemene gegevens van de PDP11/45 [5] zijn:

maximaal fysisch geheugen	248 Kb
effectieve geheugencyclus	495 nsec
maximale adresruimte	2**16 bytes
geen writable control store	
woordlengte	16 bits

Bij de PDP11/45 is de adresruimte verdeeld in maximaal 8 segmenten (DEC: Pages).

3.1.2. Benchmark resultaten.

De benchmark programma's zijn getest en uitgevoerd op de PDP11/45 configuratie van het Mathematisch Centrum met bedrijfssysteem UNIX versie 6. De PDP11/45 configuratie is als volgt:

geheugen	224 Kb (495 nsec)
cache Fabri-Tek 4511	1 Kb
FP 11 floatingpoint hardware	
RJP04 disk voor swappen	(88 Mb)

De grootte van de objectcode is als volgt:

programma	data	code
B1	504	218
B2	68	378
B3	40	208
B4	128	790

De executietijden zijn gemeten met en zonder cache:

Met cache zijn deze:

	B1	B2	B3	B4
totale tijd	52.00	45.00	58.00	68.00
system overhead	0.04	0.12	0.40	0.50
reële tijd	51.60	44.58	56.76	61.68

Zonder cache zijn deze:

	B1	B2	B3	B4
totale tijd	82.00	78.00	93.00	75.00
system overhead	0.08	0.12	0.60	0.16
reële tijd	82.00	76.80	91.14	74.44

3.2. PDP11/60.

De PDP11/60 is de kleinste van de microprogrammeerbare computers die in dit onderzoek betrokken zijn.

3.2.1. Hardware.

De PDP11/60 heeft de volgende beperkingen:

maximaal fysisch geheugen	248 Kb
effectieve geheugencyclus	550 nsec
maximale adresruimte	2**16 bytes
maximale writable control store	6 Kb
woordlengte	16 bits

De geheugenorganisatie is identiek aan die van de andere modellen uit de PDP11-serie. Dit houdt onder andere in dat de totale adresruimte in maximaal 8 segmenten (DEC: pages) verdeeld kan worden. Dit geeft onvoldoende ondersteuning om een virtueelgeheugensysteem met paginering te implementeren.

De microprogrammeringsfaciliteiten [6] zijn zeer beperkt: 48 bits microwoorden, horizontale architectuur met hardware instructiedecodering, d.w.z. dat uitsluitend instructies in PDP11 formaat gemicroprogrammeerd kunnen worden.

3.2.2. UNIX.

Het bedrijfssysteem UNIX is zonder meer beschikbaar op de PDP11/60.

3.2.3. Conclusie.

De PDP11/60 is aanzienlijk minder flexibel dan de overige twee microprogrammeerbare computers. De beperkingen van de maximale adresruimte (2*64 Kb) en de geringe mogelijkheden tot microprogrammering voldoen niet aan de basiseisen. Besloten is daarom deze computer niet verder in het onderzoek te betrekken.

3.3. VAX11/780.

3.3.1. Hardware.

De VAX11/780 heeft een MASSBUS (voor snelle i/o) en een UNIBUS [7]. De UNIBUS maakt het mogelijk dat de op het MC aanwezige randapparatuur zonder veel moeite op de VAX11/780 aangesloten kan worden. De gevens zijn als volgt:

maximaal fysisch geheugen	2 Mb
effectieve geheugencyclus	290 nsec
maximale adresruimte	2**31 bytes
maximale writable control store	12 Kb
woordlengte	32 bits

De geheugenorganisatie is geheel gericht op het ondersteunen van een virtueel-geheugensysteem met paginering. Het biedt onder andere: status- en accesinformatie per pagina, cache van page table entries om het aantal referenties naar de page table te minimaliseren.

De VAX11/780 is voorzien van drie cachesystemen: geheugencache (8 Kb), instructiecache (8) en een cache voor adresvertaling (128). Deze caches hebben een zeer gunstige invloed op de executiesnelheid van programma's.

Evenals de PDP11/60 is de VAX11/780 microprogrammeerbaar. De microwoorden zijn 96 bits groot. De instructiedecodering is flexibeler dan op de PDP11/60. DEC geeft echter geen enkele ondersteuning bij het gebruik van de microprogrammeringsfaciliteiten van de VAX11/780.

3.3.2. UNIX.

Het UNIX bedrijfssysteem van Bell Laboratories (Holmdell) voor de VAX11/780 zal in april 1979 beschikbaar zijn. Bovendien heeft Interactive Systems Inc. een implementatie van UNIX op de VAX11/780 op de markt gebracht.

3.3.3. Benchmark resultaten.

De benchmark programma's zijn tenuitvoer gebracht op de volgende VAX configuratie:

geheugen	1 Mb (600 nsec)
cache	8 Kb (200 nsec)
FP 780 floatingpoint hardware	
RP06 disk voor swappen	

Het bedrijfssysteem was VAX/VMS versie 1.0.

De benchmark programma's zijn door de leverancier op twee manieren geïmplementeerd (zie voor de programma's de appendices):

- Zo nauwgezet mogelijke vertaling van de verschafte PDP11 assembler programma's naar VAX assembler. De grootte van de objectcode is (in bytes):

programma	data	code
B1	501	154
B2	63	237
B3	76	109
B4	72	239

- Gebruikmakend van de specifieke eigenschappen van de VAX instructies, zijn twee benchmark programma's "versneld" (programma's met de toevoeging "sp"). De grootte van de objectcode is (in bytes):

programma	data	code
B1sp	501	164
B2sp	63	193

Alle programma's zijn driemaal uitgevoerd. De executietijden zijn:

Met floatingpoint accelerator:

	nr	B1	B1sp	B2	B2sp	B3	B4
totale	1	28	5	22	6	51	14
tijd	2	28	5	22	6	51	14
	3	28	5	23	6	51	14
system	1	0.5	0.1	0.4	0.1	0.8	0.2
overhead	2	0.3	0.1	0.3	0.1	0.7	0.2
	3	0.5	0.1	0.3	0.1	0.8	0.3
reële	1	27.4	4.6	21.8	5.7	49.5	13.1
tijd	2	27.4	4.6	21.9	5.7	49.5	13.1
	3	27.3	4.6	21.8	5.7	49.5	13.1

Zonder floating point accelerator:

	nr	B1	B1sp	B2	B2sp	B3	B4
totale	1	29	5	22	6	51	50
tijd	2	28	5	23	6	50	49
	3	28	5	22	6	50	50
system	1	0.5	0.2	0.4	0.2	0.8	0.6
overhead	2	0.5	0.2	0.4	0.1	0.8	0.6
	3	0.4	0.2	0.3	0.1	0.7	0.6
reële	1	27.3	4.6	21.9	5.7	49.5	48.5
tijd	2	27.3	4.6	21.9	5.7	49.5	48.5
	3	27.4	4.6	21.9	5.7	49.5	48.5

3.3.4. Conclusie.

De VAX11/780 voldoet aan alle eisen zeer goed. Microprogrammeren op de VAX11/780 is niet eenvoudig, ook omdat de ondersteuning van de fabrikant geheel ontbreekt. Het instructierepertoire leidt tot compacte efficiënte code, zonder dat hiervoor verregaande optimalisaties uitgevoerd moeten worden. De hardware compatibiliteit (UNIBUS) maakt de machine erg aantrekkelijk.

3.4. Interdata 8/32.

3.4.1. Hardware.

Voor de interdata 8/32 [8] gelden de volgende maxima:

maximaal physisch geheugen	1Mb
effectieve geheugencyclus	300 ns
maximale adresruimte	2**24 bytes
maximale writable control store	16Kb
woordlengte	32 bits

De interdata 8/32 biedt evenals de PDP11/60 geheugensegmentatie. De totale adresruimte kan onderverdeeld worden in maximaal 16 segmenten van maximaal 64 Kb elk. Per segment wordt informatie over status en soort acces bijgehouden. De interdata 8/32 heeft, in tegenstelling tot de PDP11/60 en VAX11/780, een verticale microinstructieset. Hierdoor is deze machine aanzienlijk eenvoudiger te microprogrammeren. Bovendien wordt microprogrammering door de fabrikant zeer goed ondersteund: uitgebreide documentatie (zie [9], [10] en [11]) en programmatuur voor de ontwikkeling van microprogramma's (microassembler, microdebugger). Geen van beide wordt door DEC geleverd.

De instructieset van de 8/32 laat nogal wat te wensen over. Veel operaties (arithmetisch, logisch) zijn niet beschikbaar in alle instructieformaten voor alle mogelijke datatypen. Een codegenerator voor deze machine zal hierdoor rekening moeten houden met veel speciale gevallen.

3.4.2. UNIX.

Het UNIX systeem is direct beschikbaar op de 8/32. De universiteit van Wollongong, Australie, heeft UNIX zowel op de de interdata 7/32 als op de interdata 8/32 geïmplementeerd.

Dankzij de welwillende medewerking van de afdeling bouwkunde van de Technische Hogeschool Eindhoven, waren wij in staat om het UNIX systeem op de daar aanwezige 8/32 te installeren en te bestuderen. Wij hebben hierbij zeer veel hulp en welwillendheid ondervonden van vertegenwoordigers van Perkin-Elmer.

De algemene indruk is dat het hier een op gebruikersniveau volledig compatibele implementatie van UNIX versie 6 betreft. Een aanzienlijke hoeveelheid systeemprogramma's is op de 8/32 beschikbaar. Naast de resultaten van de benchmarks (zie volgende paragraaf), is bovendien een verdergaande vergelijking tussen een PDP11/45 met cache en 2 RK05 schijven en interdata 8/32 gemaakt. De onderstaande tabel bevat snelheidsverhoudingen. Een waarde X wil zeggen: in de betreffende meting is de Interdata 8/32 X maal zo snel als de PDP11/45. De resultaten zijn:

copie maken van een file op disk:

totale tijd:	2.05
reële tijd:	1.05
systeemoverhead:	1.33

C compilaties:

totale tijd:	1.69
reële tijd:	0.96
systeemoverhead:	1.53

Bovendien werd de grootte van de objectcode (zonder data) gemeten voor een vijftigtal C programma's. Hierbij bleek de pure code van 8/32 programma's circa 1.6 maal zo groot te zijn als de pure code van overeenkomstige PDP11 programma's. Men dient op te merken dat hierbij zowel eigenschappen van de 8/32 instructieset als eigenschappen van de 8/32 C compiler een rol spelen.

3.4.3. Benchmark resultaten.

De door de fabrikant geleverde resultaten zien er als volgt uit:

	normal (UNIX)	recoded (OS)
B1	122	98.9
B2	52.9	8.5
B3	76	51.1
B4	22.2	10.3

Deze gegevens zijn helaas vrij onvolledig. Wij hebben zelf ook enige metingen aan deze programma's verricht; deze cijfers zijn in de samenvattende tabel in sectie 3.5. opgenomen (deze vermeldingen staan daar tussen "[]"-haken).

3.4.4. Conclusie.

De interdata 8/32 voldoet redelijk aan de gestelde eisen, hoewel de resultaten van de benchmarks teleurstellend zijn. Het vrij asymmetrische instructierepertoire moet als nadeel worden aangemerkt voor een researchcomputer. De mogelijkheden voor microprogrammering zijn echter aantrekkelijk.

3.5. Samenvatting.

De belangrijkste cijfers van de vergelijking tussen de machines zijn verwerkt in een tabel. Tussen "()"-haken staan de cijfers vermeld verkregen zonder de versnellende hardware (cache voor de PDP11/45 en floatingpoint accelerator voor de VAX11/780). Tussen "[]"-haken staan de cijfers vermeld zoals verkregen door benchmark metingen verricht door het Mathematisch Centrum op de Interdata 8/32.

computer	progr.	data bytes	code bytes	totale tijd sec.		reële tijd sec.	
PDP11/45	B1	504	218	52.0	(82.0)	51.6	(82.0)
VAX11/780	B1	501	154	28	(28)	27.4	(27.3)
VAX11/780	B1sp	501	164	5	(5)	4.6	(4.6)
Interdata8/32	B1	[504]	[324]	122	[127]		[124.90]
Interdata8/32	B1sp			98.9			
PDP11/45	B2	68	378	45.0	(78.0)	44.6	(96.8)
VAX11/780	B2	63	237	22	(22)	21.8	(21.9)
VAX11/780	B2sp	63	193	6	(6)	5.7	(5.7)
Interdata8/32	B2	[132]	[488]	52.9	[51]		[49.98]
Interdata8/32	B2sp			8.5			
PDP11/45	B3	40	208	58.0	(78.0)	56.8	(91.1)
VAX11/780	B3	76	109	51	(50)	49.5	(49.5)
Interdata8/32	B3			76			
Interdata8/32	B3sp			51.1			
PDP11/45	B4	128	790	60.0	(93.0)	60.4	(74.4)
VAX11/780	B4	72	239	14	(50)	13.1	(48.5)
Interdata8/32	B4			22.2			
Interdata8/32	B4sp			10.3			

Referenties

[1]
Aanvraag van een Research-machine t.b.v. Afd. Informatika van de Katholieke Universiteit Nijmegen, P. Holager, C.H.A. Koster, M. Stahl en H.J. Thomassen, juni 1978.

[2]
Wetenschappelijk programma 1979, Meerjarenplanning 1980-85, Mathematische Centrum.

[3]
The UNIX Time-Sharing System, Dennis M. Ritchie, Ken Thompson,
Communications of the ACM, 17(1974)7, 365-375

[4]
C Reference Manual, Dennis M. Ritchie, Bell Telephone Laboratories, Murray Hill, New Jersey, 1974.

[5]
PDP11/45 Processor Handbook, Digital Equipment Corporation, 1974.

[6]
PDP11/60 Processors Handbook, Digital Equipment Corporation, 1977.

[7]
VAX11/780 Architecture Handbook vol. 1, Digital Equipment Corp., 1977/78.

[8]
Model 8/32 processor user's manual, Interdata Inc., 1975.

[9]
Model 8/32 micro-instruction reference manual, Interdata Inc., 1975.

[10]
Interdata 8/32 writable control store (wcs) user's manual, Interdata Inc., 1975.

[11]
Common microcode assembler language (microcal) user's manual, Interdata Inc., 1975.

Appendix I Benchmark 1 C-programma

```
#
/*
 * Benchmark #1 -- byte handling
 */

#define NTIMES  40
#define NCHARS  500

char a[NCHARS+1];

/*
 * rotate - rotates the characters in "a" one place to the left.
 * - "s" and "t" are character pointers.
 * - the notation "*s++" means:
 *     . fetch the character to which "s" points and
 *     . increment "s", i.e. let "s" point to the next character
 */

rotate(){
        register char first, *s, *t;

        s = t = a;
        first = *s++;
        while(*s != '\0')
                *t++ = *s++;
        *t++ = first;
}

/*
 * check -- make sure that array "a" contains "xaaa...aaa".
 */

check(){
        register int i;
        register char *s;

        s = a;
        if(*s++ != 'x')
                abort();
        while(*s)
                if(*s++ != 'a')
                        abort();
}

/*
 * Main program
 * - fill character array "a" with "xaaa...aaa" followed by
 *   a null character ('\0') as string terminator.
 * - rotate "a" NCHARS times one place to the left.
 *   after each rotation, ensure that the "x" character is in
 *   the right position.
 * - after NCHARS rotations check that "a" still contains
 *   "xaaa...aaa".
 */
```

```
main(){
        register int i, n;

        for(n = 0; n < NTIMES; n++){
                a[0] = 'x';
                for(i = 1; i < NCHARS; i++)
                        a[i] = 'a';
                a[NCHARS] = '\0';
                check();
                for(i = 0; i < NCHARS; i++){
                        rotate();
                        if(a[NCHARS-i-1] != 'x')
                                abort();
                }
                check();
        }
}
```

Appendix II Benchmark 2 C-programma

```
#
/*
 * Benchmark #2 -- bit handling
 */

#define NTIMES  2               /* number of iterations */
#define B       4
#define BPW     (1<<B)          /* bits per word */
#define BPOS    ((1<<B)-1)      /* mask for bit position
                                   in word */
#define NBITS   500             /* size of bit array */

int a[(NBITS+2*BPW)/BPW];

/*
 * fetch -- returns bit #n in bitarray "a".
 * - bit addresses are computed as follows:
 *   . the higher order BPW-B bits are the address of the
 *     word in which a bit occurs.
 *   . the lower order B bits are the bit position in that word.
 *   . a[n >> B] is the desired word
 *   . n & BPOS is the bit position
 *   . 1 << (n & BPOS) is bit pattern with a "1" on the required
 *     bit position and "0" otherwise.
 *   . a[n >> B] & (1 << (n & BPOS)) is zero if the required bit
 *     is "0" and non-zero otherwise.
 * - e ? a : b is a conditional expression which has as value:
 *   if expression "e" is non-zero then "a" and "b" otherwise.
 * - this program uses the following logical operators:
 *   >> right shift
 *   << left shift
 *   &  (bitwise) and
 *   |  (bitwise) or
 *   ~  (bitwise) complement
 */

int fetch(an)
int an;{
        register int n;

        n = an;
        return(
                (a[n >> B] & (1 << (n & BPOS))) ? 1 : 0
        );
}

/*
 * store -- stores value "v" in bit #n of bitarray "a".
 */

store(an, v)
int an, v;{
        register int bpos, n;
```

```
        n = an;
        bpos = (n & BPOS);
        a[n >> B] = (a[n >> B] & ~(1 << bpos)) | (v << bpos);
}

/*
 * rotate -- rotates bitarray "a" one bit position to the left.
 */

rotate(){
        register int first, i;

        first = fetch(0);
        for(i = 0; i < NBITS - 1; i++)
                store(i, fetch(i + 1));
        store(NBITS - 1, first);
}

/*
 * check -- ensure that bitarray "a" contains "01111...111".
 */

check(){
        register int i;

        if(fetch(0) != 0)
                abort();
        for(i = 1; i < NBITS; i++)
                if(fetch(i) != 1)
                        abort();
}

/*
 * Main program.
 * - fill array "a" with the bit string "011...111".
 * - check that "a" contains this bit string.
 * - rotate "a" one bit position to the left and
 *   make sure that the "0" bit is in the right position.
 * - after NBITS rotations check that "a" contains
 *   the value "011...111".
 */

main(){
        register int i, n;

        for(n = 0; n < NTIMES; n++){
                store(0, 0);
                for(i = 1; i < NBITS; i++)
                        store(i, 1);
                check();
                for(i = 0; i < NBITS; i++){
                        rotate();
                        if(fetch(NBITS - i - 1) != 0)
                                abort();
                }
                check();
```

```
        }
}
```

Appendix III Benchmark 3 C-programma

```
#
/*
 * Benchmark #3 -- procedure calls
 */

#define NTIMES  2

/*
 * tab -- table with some values of Ackermann's function.
 */
int tab [] {
/*      i,      j,      ack(i, j) */
        0,      0,      1,
        0,      1,      2,
        2,      0,      3,
        3,      3,      61,
        3,      5,      253,
        3,      7,      1021,
        -1
};

/*
 * ack -- compute Ackermann's function
 * Ackermann's function is a highly recursive function that
 * heavily exercizes the procedure calling mechanism.
 * Here occurs a nested conditional expression with the meaning
 *  if m equals 0 then return n + 1
 *  else if n equals 0 then return ack(m-1,1)
 *       else return ack(m-1,ack(m,n-1))
 */
int ack(m,n)
int m,n;{
        return( (m == 0) ? (n + 1)
                         : ((n == 0) ? ack(m-1,1)
                                     : ack(m-1, ack(m,n-1))
                           )
                );
}

/*
 * Main program.
 * - for all table values, call ack with the arguments in the
 *   table and compare the function value with the value
 *   in the table.
 */
main(){
        register int i, n;

        for(n = 0; n < NTIMES; n++){
                for(i = 0; tab[i] >= 0; i =+ 3)
                        if(ack(tab[i], tab[i + 1]) != tab[i + 2])
                                abort();
        }
}
```

Appendix IV Benchmark 4 C-programma

```
#
/*
 * Benchmark #4 -- floating point
 */

#define NTIMES  1000

double x, y, x1, y1, m10, m11;
double m00  0.999847695156391;  /* cos 1 */
double m01  0.017452406437284;  /* sin 1 */
double eps 0.00001;

/*
 * diff -- compare two floating point numbers with tolerance "e".
 *         returns 0 for within tolerance
 *         returns 1 for different.
 */

int diff(xa, xb, e)
float xa, xb, e;{
        if(xa > xb)
                return(xa - xb > e);
        else
                return(xb - xa > e);
}

/*
 * Main program.
 * - define a 2x2 matrix that describes a rotation around
 *   1 degree.
 * - next apply this matrix to the vector (1.0, 0.0)
 * - after 360 rotations this vector should have been rotated
 *   to its original position; check that this is the case
 *   within tolerance "eps".
 */

main(){
        register int i, n;

        /*
         * Define the rotation matrix
         *       cos 1  sin 1
         *      -sin 1  cos 1
         */
        m10 = - m01;
        m11 = m00;

        for(n = 0; n < NTIMES; n++){
                x = 1.0; y = 0.0;

                for(i = 0; i < 360; i++){
                        x1 = m00 * x + m01 * y;
                        y1 = m10 * x + m11 * y;
                        x = x1;
```

```
                y = y1;
        }
        if(diff(x, 1.0, eps))
                abort();
        if(diff(y, 0.0, eps))
                abort();
    }
}
```

Appendix V Procedure ingang- en uitgangmechanisme

```
//
// rts -- provide C runtime system and
//        operating system interface
// Common header for all benchmark programs:
// Program execution starts at label "start".
// Additional routines:
//      csv     register save routine called on procedure entry
//      cret    label jumped to on procedure exit
//      abort   procedure to abort program execution
//              in an error situation
//

.globl  _main,csv,cret,_abort

start:
        setd                    / set double precision
                                / floating point
        jsr     pc,_main        / call main program
        sys     exit            / exit to operating system

// C register save and restore -- version 12/74
// See the document C.calling for more details concerning
// the C calling sequence and layout of stack frames.
//

csv:
        mov     r5,r0
        mov     sp,r5
        mov     r4,-(sp)
        mov     r3,-(sp)
        mov     r2,-(sp)
        tst     -(sp)
        jmp     (r0)

cret:
        mov     r5,r1
        mov     -(r1),r4
        mov     -(r1),r3
        mov     -(r1),r2
        mov     r5,sp
        mov     (sp)+,r5
        rts     pc

iot     = 4

_abort:
        iot
```

Appendix VI Benchmark 1 PDP11 assembler programma

```
//
// Benchmark #1 -- byte handling
//
NCHARS  = 500.
NTIMES  = 40.
SIZE    = [NCHARS+1]
_a:     .=.+SIZE
.even
//
// rotate
//
.text
..first=r4
..s=r3
..t=r2
_rotate:
        jsr     r5,csv          / save registers
        mov     $_a,..t         / t = a
        mov     ..t,..s         / s = t
        movb    (..s)+,..first  / first = *s++
        jbr     2f
1:      movb    (..s)+,(..t)+   / *s++ = *t++
2:      tstb    (..s)
        jne     1b              / *s != '\0' ?
        movb    ..first,(..t)+  / *t++ = first
        jmp     cret            / return
//
// _check
//
..i=r4
..s=r3
_check:
        jsr     r5,csv          / save registers
        mov     $_a,..s         / s = a
        cmpb    $'x,(..s)+
        jeq     2f              / *s++ == 'x' ?
1:      jsr     pc,_abort
2:      tstb    (..s)
        jne     3f              / *s != '\0' ?
        jmp     cret
3:      cmpb    $'a,(..s)+
        jeq     2b              / *s++ == 'a' ?
        jbr     1b
//
// _main
//
..i=r4
..n=r3
.globl  _main
_main:
        jsr     r5,csv          / save registers
        clr     ..n
1:      movb    $'x,_a          / a[0] = 'x'
        mov     $1,..i          / i = 1
```

```
2:      movb    $'a,_a(..i)     / a[i] = 'a'
        inc     ..i             / i = i + 1
        cmp     $NCHARS,..i
        jgt     2b              / NCHARS > i ?
        clrb    NCHARS+_a       / a[NCHARS] = '0'
        jsr     pc,_check       / check()
        clr     ..i             / i = 0
3:      jsr     pc,_rotate      / rotate()
        mov     $NCHARS,r0
        sub     ..i,r0          / r0 = NCHARS - i
        cmpb    $'x,-1+_a(r0)
        jeq     4f              / a[NCHARS - i + 1] == 'x' ?
        jsr     pc,_abort       / abort()
4:      inc     ..i             / i = i + 1
        cmp     $NCHARS,..i
        jgt     3b              / NCHARS > i ?
        jsr     pc,_check       / check()
        inc     ..n             / n = n + 1
        cmp     $NTIMES,..n
        jgt     1b              / NTIMES > n ?
        jmp     cret            / return
```

Appendix VII Benchmark 2 PDP11 assembler programma

```
//
// Benchmark #2 -- bit handling
//
NTIMES  = 2.
NBITS   = 500.
B       = 4
BPW     = [1\<B]                    / 1 << B
BPOS    = ![[1\<B]-1]               / ((1 << B) - 1)
SIZE    = 2*[[NBITS+[2*BPW]]\/BPW]
_a:     .=.+SIZE
.even
.text
//
// fetch
//
..n=r4
..an=4
_fetch:
        jsr     r5,csv              / save registers
        mov     ..an(r5),..n        / n = an
        mov     $1,r0               / r0 = 1
        mov     ..n,r1              / r1 = n
        bic     $BPOS,r1            / r1 = n  BPOS
        ash     r1,r0               / r0 = 1 << (n  BPOS)
        mov     ..n,r1              / r1 = n
        ash     $-B,r1              / r1 = n >> B
        asl     r1                  / turn word address
                                    / in byte address
        mov     _a(r1),r1           / r1 = a[n >> B]
        bit     r1,r0
        jeq     2f                  / r1  (1 << (n  BPOS)) ?
        mov     $1,r0               / 1 return value
1:      jmp     cret                / return
2:      clr     r0                  / 0 return value
        jbr     1b
//
// store
//
..n=r3
..v=6
..bpos=r4
..an=4
_store:
        jsr     r5,csv              / save registers
        mov     ..an(r5),..n        / n = an
        mov     ..n,..bpos          / bpos = n
        bic     $BPOS,..bpos        / bpos = bpos & BPOS
        mov     ..n,r0              / r0 = n
        ash     $-B,r0              / r0 = n >> B
        asl     r0                  / turn word address
                                    / in to byte address
        mov     ..n,r1              / r1 = n
        ash     $-B,r1              / r1 = n >> B
        asl     r1                  / turn word address
```

```
                                / in to byte address
        mov     _a(r1),r1       / r1 = a[n >> B]
        mov     $1,r2           / r2 = 1
        ash     ..bpos,r2       / r2 = 1 << bpos
        bic     r2,r1           / r1 = a[n >> B] & ~(1 << bpos)
        mov     ..v(r5),r2      / r2 = v
        ash     ..bpos,r2       / r2 = v << bpos
        bis     r2,r1           / r1 = (a[n >> B]  ~(1 << bpos))
                                / | (v << bpos)
        mov     r1,_a(r0)
        jmp     cret
//
// rotate
//
..first=r4
..i=r3
_rotate:
        jsr     r5,csv          / save registers
        clr     (sp)
        jsr     pc,*$_fetch     / fetch(0)
        mov     r0,..first      / first = fetch(0)
        clr     ..i             / i = 0
1:      mov     ..i,(sp)
        inc     (sp)
        jsr     pc,*$_fetch     / fetch(i + 1)
        mov     r0,(sp)
        mov     ..i,-(sp)
        jsr     pc,*$_store     / store(i, fetch(i + 1))
        tst     (sp)+           / remove args
        inc     ..i             / i = i + 1
        cmp     $[NBITS-1],..i
        jgt     1b              / NBITS - 1 > i ?
        mov     ..first,(sp)
        mov     $[NBITS-1],-(sp)
        jsr     pc,*$_store     / store(NBITS - 1, first)
        tst     (sp)+           / remove args
        jmp     cret            / return
//
// check
//
..i=r4
_check:
        jsr     r5,csv          / save registers
        clr     (sp)
        jsr     pc,*$_fetch     / fetch(0)
        tst     r0
        jeq     1f              / fetch(0) == 0 ?
        jsr     pc,_abort       / abort()
1:      mov     $1,..i          / i = 1
2:      mov     ..i,(sp)
        jsr     pc,*$_fetch     / fetch(i)
        cmp     $1,r0
        jeq     3f              / fetch(i) == 1 ?
        jsr     pc,_abort       / abort()
3:      inc     ..i             / i = i + 1
        cmp     $NBITS,..i
```

```
        jgt     2b              / NBITS > i ?
        jmp     cret            / return
//
// main
//
..i=r4
..n=r3
.globl  _main
_main:
        jsr     r5,csv          / save registers
        clr     ..n             / n = 0
1:      clr     (sp)
        clr     -(sp)
        jsr     pc,*$_store     / store(0, 0)
        tst     (sp)+           / remove args
        mov     $1,..i          / i = 1
2:      mov     $1,(sp)
        mov     ..i,-(sp)
        jsr     pc,*$_store     / store(i, 1)
        tst     (sp)+           / remove args
        inc     ..i             / i = i + 1
        cmp     $NBITS,..i
        jgt     2b              / NBITS > i ?
        jsr     pc,_check       / check()
        clr     ..i             / i = 0
3:      jsr     pc,_rotate      / rotate()
        mov     $NBITS,(sp)
        sub     ..i,(sp)
        dec     (sp)
        jsr     pc,*$_fetch     / fetch(NBITS - i - 1)
        tst     r0
        jeq     4f              / fetch(NBITS - i - 1) == 0 ?
        jsr     pc,_abort       / abort()
4:      inc     ..i             / i = i + 1
        cmp     $NBITS,..i
        jgt     3b              / NBITS > i ?
        jsr     pc,_check       / check()
        inc     ..n             / n = n + 1
        cmp     $NTIMES,..n
        jgt     1b              / NTIMES > n ?
        jmp     cret            / return
```

Appendix VIII Benchmark 3 PDP11 assembler programma

```
//
// Benchmark #3 -- procedure calls
//
NTIMES = 2.
.data
_tab:
0.;     0.;     1.
0.;     1.;     2.
2.;     0.;     3.
3.;     3.;     61.
3.;     5.;     253.
3.;     7.;     1021.
-1.
.text
//
// ack
//
..m=4
..n=6
_ack:
        jsr     r5,csv          / save registers
        tst     ..m(r5)
        jne     4f              / m != 0 ?
        mov     ..n(r5),r0
        inc     r0              / r0 = n + 1
        jbr     3f
1:      mov     $1,(sp)
2:      mov     ..m(r5),-(sp)
        dec     (sp)
        jsr     pc,*$_ack       / ack(m - 1, 1)
        tst     (sp)+           / remove args
3:      jmp     cret            / return
4:      tst     ..n(r5)
        jeq     1b              / n == 0 ?
        mov     ..n(r5),(sp)
        dec     (sp)
        mov     ..m(r5),-(sp)
        jsr     pc,*$_ack       / ack(m , n - 1)
        tst     (sp)+           / remove args
        mov     r0,(sp)
        jbr     2b              / join with code for ack(m-1,1)
                                / with 1 replaced by ack(m,n-1)
//
// main
//
..i=r4
..n=r3
.globl  _main
_main:
        jsr     r5,csv          / save registers
        clr     ..n             / n = 0
1:      clr     ..i             / i = 0
        jbr     4f
2:      mov     ..i,r0
```

```
        asl     r0
        mov     2+_tab(r0),(sp) / tab[i+1]
        mov     ..i,r0
        asl     r0
        mov     _tab(r0),-(sp)  / tab[i]
        jsr     pc,*$_ack       / ack(tab[i], tab[i+1])
        tst     (sp)+           / remove args
        mov     ..i,r1
        asl     r1
        cmp     4+_tab(r1),r0
        jeq     3f              / tab[i+2] ==
                                /  ack(tab[i],tab[i+1]) ?
        jsr     pc,_abort       / abort()
3:      add     $3,..i          / i = i + 3
4:      mov     ..i,r0
        asl     r0
        tst     _tab(r0)
        jge     2b              / tab[i] >= 0 ?
        inc     ..n             / n = n + 1
        cmp     $NTIMES,..n
        jgt     1b              / NTIMES > n ?
        jmp     cret            / return
```

Appendix IX Benchmark 4 PDP11 assembler programma

```
//
// Benchmark #4 -- floating point
//
NTIMES = 1000.
.ONE    = 40200          / 1.0
.data
_x:     .=.+10
_y:     .=.+10
_x1:    .=.+10
_y1:    .=.+10
_m10:   .=.+10
_m11:   .=.+10
_m00:   40177;173004;137655;76075       / 0.999847695156391
_m01:   36616;174131;56116;166526       / 0.017452406437284
_eps:   34047;142654;43433;43604        / 0.00001
.text
//
// diff
//
..e=24
..xa=4
..xb=14
_diff:
        jsr     r5,csv          / save registers
        movf    ..xa(r5),r0
        cmpf    ..xb(r5),r0
        cfcc
        jge     2f              / xb >= xa ?
        subf    ..xb(r5),r0
        cmpf    ..e(r5),r0
        cfcc
        jlt     3f              / eps < xa - xb ?
1:      clr     r0              / 0 return value
        jbr     4f
2:      movf    ..xb(r5),r0
        subf    ..xa(r5),r0
        cmpf    ..e(r5),r0
        cfcc
        jge     1b              / eps >= xb - xa ?
3:      mov     $1,r0           / 1 return value
4:      jmp     cret
//
// main
//
..i=r4
..n=r3
_main:
        jsr     r5,csv          / save registers
        movf    _m01,r0
        negf    r0
        movf    r0,_m10         / m10 = -m01
        movf    _m00,r0
        movf    r0,_m11         / m11 = m00
        clr     ..n             / n = 0
```

```
1:      movf    $.ONE,r0
        movf    r0,_x           / x = 1.0
        clrf    _y              / y = 0.0
        clr     ..i             / i = 0
2:      movf    _m00,r0
        mulf    _x,r0           / r0 = m00 * x
        movf    _m01,r1
        mulf    _y,r1           / r1 = m01 * y
        addf    r1,r0
        movf    r0,_x1          / x1 = m00 * x + m01 * y
        movf    _m10,r0
        mulf    _x,r0           / r0 = m10 * x
        movf    _m11,r1
        mulf    _y,r1           / r1 = m11 * y
        addf    r1,r0
        movf    r0,_y1          / y1 = m10 * x + m11 * y
        movf    _x1,r0
        movf    r0,_x           / x = x1
        movf    _y1,r0
        movf    r0,_y           / y = y1
        inc     ..i             / i = i + 1
        cmp     $360.,..i
        jgt     2b              / 360 > i ?
        movf    _eps,r0
        movf    r0,-(sp)
        movf    $.ONE,r0
        movf    r0,-(sp)
        movf    _x,r0
        movf    r0,-(sp)
        jsr     pc,_diff        / diff(x, 1.0, eps)
        add     $30,sp          / remove parameters
        tst     r0
        jeq     3f              / diff == 0 ?
        jsr     pc,_abort
3:      movf    _eps,r0
        movf    r0,-(sp)
        clrf    -(sp)
        movf    _y,r0
        movf    r0,-(sp)
        jsr     pc,_diff        / diff(y, 0.0, eps)
        add     $30,sp          / remove parameters
        tst     r0
        jeq     4f              / diff == 0 ?
        jsr     pc,_abort
4:      inc     ..n             / n = n + 1
        cmp     $NTIMES,..n
        jgt     1b              / NTIMES > n ?
        jmp     cret
```

Appendix X Benchmark 1 VAX11 assembler programma

```
        .title  rts

        .entry  start,0

        calls   #0,timrb        ;start timer
        calls   #0,_main        ;call main program
        calls   #0,timre        ;stop timer and print values
        $exit_s #ss$_normal     ;normal, succesfull completion

_abort::calls   #0,timre        ;stop timer
        $exit_s #ss$_ssfail     ;fatal error

        .end    start

        .title  b1

        .psect  b1$$data,noexe,page

nchars=500                              ;size of char array
ntimes=10                               ;number of times to loop
size=nchars+1
_a:.=.+size                             ;reserve space

        .psect  b1$$code,nowrt,page

; _main

;..i=r7
;..n=r6

        .entry  _main,^m<r8,r7,r6,r5,r4,r3,r2>

        movl    #nchars,r8              ;get size of array
        clrl    r6                      ;clear loop counter
1$:     movb    #^a/x/,_a               ;put 'x' in first byte
        movc5   #0,_a,#^a/a,#nchars-1,_a+1 ;fill rest with 'a'
        clrb    _a(r8)                  ;terminate with 'null'
        calls   #0,_check               ;check setup
        clrl    r7                      ;clear char counter
3$:     calls   #0,_rotate              ;rotate one byte
        subl3   r7,r8,r0                ;calc position of 'x'
        cmpb    #^a/x/,_a-1(r0)         ;'x' in the right place ?
        beql    4$                      ;if eql yes
        jsb     _abort                  ;no, abort
4$:     aoblss  r8,r7,3$                ;loop nchars times
        calls   #0,_check               ;check char array
        aoblss  #ntimes,r6,1$           ;loop ntimes times
        ret                             ;return to caller

; _rotate

;..first=r4
;..s=r3
;..t=r2
```

```
        .entry  _rotate,^m<r5,r4,r3,r2>

        movab   _a,r2                   ;get address of _a
        movl    r2,r3                   ;copy
        movb    (r3)+,r4                ;save first byte
        brb     2$
1$:     movb    (r3)+,(r2)+             ;copy next char
2$:     tstb    (r3)                    ;at end of array ?
        bneo    1$                      ;if neq no
        movb    r4,(r2)+                ;restore byte at end
        ret                             ;return to caller

; _check

;..s=r3

        .entry  _check,^m<r5,r4,r3,r2>

        movab   -a,r3                   ;get address of _a
        cmpb    #^a/x/,(r3)+            ;is first char an 'x' ?
        beql    2$                      ;if eql yes, ok
1$:     jsb     _abort                  ;go away
2$:     tstb    (r3)                    ;at end ?
        bneq    3$                      ;if neq no
        ret                             ;return to caller
3$:     cmpb    #^a/a/,(r3)+            ;is next char an 'a' ?
        beql    2$                      ;if eql yes, ok
        brb     1$                      ;no, abort

        .end
```

Appendix XI Benchmark 1 VAX11 hercodering

```
        .title  b1s

        .psect  b1s$$data,noexe,page

nchars=500                              ;size of char array
ntimes=40                               ;number of times to loop
size=chars+1

_a:.=.+size                             ;reserve space

        .psect  b1s$$code,nowrt,page

; _main

;..i=r7
;..n=r6

        .entry  _main,^m<r8,r7,r6,r5,r4,r3,r2>

        movl    #nchars,r8              ;get size of array
        clrl    r6                      ;clear loop counter
1$:     movb    #^a/x/,_a               ;put 'x' in first byte
        movc5   #0,_a,#^a/a/,#nchars-1,_a+1 ;fill rest with 'a'
        clrb    _a(r8)                  ;terminate with 'null'
        pushl   r8                      ;push size of array
        calls   #1,_check               ;check setup
        clrl    r7                      ;clear char counter
3$:     pushl   r8                      ;push arrat size
        calls   #1,_rotate              ;rotate one byte
        subl3   r7,r8,r0                ;calc position of 'x'
        cmpb    #^a/x/,_a-1(r0)         ;'x' in the right place ?
        beql    4$                      ;if eql yes
        jsb     _abort                  ;no, abort
4$:     aoblss  r8,r7,3$                ;loop nchars times
        pushl   r8                      ;push array size
        calls   #1,_check               ;check char array
        aoblss  #ntimes,r6,1$           ;loop ntimes times
        ret

; _rotate

;..first=r8
;..s=r7
;..t=r6

        .entry  _rotate,^m<r8,r7,r6,r5,r4,r3,r2>

        movab   _a,r6                   ;get address of _a
        movl    r6,r7                   ;copy
        movb    (r7)+,r8                ;save first byte
        subl3   #1,4(ap),r5             ;get size of array minus one
        movc3   r5,(r7),(r6)            ;shift array to left
        movb    r8,(r3)+                ;restore byte at end
        ret                             ;return to caller
```

```
; _check

;..s=r6

        .entry  _check,^m<r6,r5,r4,r3,r2>

        movab   _a,r6                   ;get address of _a
        cmpb    #^a/x/,(r6)+            ;is first char an 'x' ?
        beql    2$                      ;if eql yes, ok
1$:     jsb     _abort                  ;go away
2$:     subl3   #1,4(ap),r5             ;size of array minus one
        skpc    #^a/a/,r5,(r6)          ;skip all 'a's
        bneq    1$                      ;if neq found difference
        tstb    (r1)                    ;is next null ?
        bneq    1$                      ;if neq no, abort
        ret                             ;return to caller

        .end
```

Appendix XII Benchmark 2 VAX11 assembler programma

```
        .title  b2

        .psect  b2$$data,noexe,page

nbits=500                               ;size of bit array
ntimes=2                                ;times to loop
_a:.=.+<<500+8>/8>                      ;reserve space

        .psect  b2$$code,nowrt,page

; _main

;..i=r4
;..n=r3

        .entry  _main,^m<r5,r4,r3,r2>

        movl    #nbits,r2               ;get size of bitarray
        clrl    r3                      ;clear loop counter
1$:     clrq    -(sp)                   ;push value zero and pos zero
        calls   #2,_store               ;store
        movl    #1,r4                   ;set pos to 1
2$:     pushl   #1                      ;push value 1
        pushl   r4                      ;push position
        calls   #2,_store               ;store
        aoblss  r2,r4,2$                ;loop through array
        calls   #0,_check               ;check the array
        clrl    r4                      ;clear bits counter
3$:     calls   #0,_rotate              ;rotate the array
        subl3   r4,#nbits-1,-(sp)       ;check the zero bit
        calls   #1,_fetch               ;get it
        tstl    r0                      ;is it really zero ?
        beql    4$                      ;if eql yes, hurray !
        jsb     _abort                  ;no, die
4$:     aoblss  r2,r4,3$                ;do a complete loop
        calls   #0,_check               ;check array
        aoblss  #ntimes,r3,1$           ;loop ntimes times
        ret                             ;return to caller

; _rotate

;..first=r4
;..i=r3

        .entry  _rotate,^m<r5,r4,r3,r2>

        clrl    -(sp)                   ;bit zero
        calls   #1,_fetch               ;get it
        movl    r0,r4                   ;save it
        clrl    r3                      ;clear counter
1$:     addl3   r3,#1,_(sp)             ;push bit number
        calls   #1,$fetch               ;get bit
        pushl   r0                      ;push bit
        pushl   r3                      ;push bit number
```

```
        calls   #2,_store               ;save bit
        aoblss  #nbits-1,r3,1$          ;loop thru bitarray
        pushl   r4                      ;push first bit
        pushl   #nbits-1                ;push address of last bit
        calls   #2,_store               ;store it
        ret                             ;return to caller

; _check

;..i=r4

        .entry  _check,^m<r5,r4,r3,r2>

        clrl    -(sp)                   ;bit zero
        calls   #1,_fetch               ;get it
        tstl    r0                      ;is it zero ?
        beql    1$                      ;if eql yes, ok
        jsb     _abort                  ;go away
1$:     movl    #1,r4                   ;init counter
2$:     pushl   r4                      ;push bit number
        calls   #1,_fetch               ;get bit
        cmpb    #1,r0                   ;is it one ?
        beql    3$                      ;if eql yes
        jsb     _abort                  ;no, terminate
3$:     aoblss  #nbits,r4,2$            ;loop thru array
        ret                             ;return to caller

; _fetch

..an=4

        .entry  _fetch,0

        extzv   ..an(ap),#1,_a,r0       ;extract bit
        ret                             ;return to caller

; _store

..an=4
..v=8

        .entry  _store,0

        insv    ..v(ap),..an(ap),#1,_a  ;insert bit
        ret                             ;return to caller

        .end
```

Appendix XIII Benchmark 2 VAX11 hercodering

```
        .title  b2s

        .psect  b2s$$data,noexe,page

nbits=500                               ;size of bit array
ntimes=2                                ;times to loop
_a:.=.+<<500+8>/8>                      ;reserve space

        .psect  b2s$$code,nowrt,page

; _main

;..i=r4
;..n=r3

        .entry  _main,^m<r5,r4,r3,r2>

        movl    #nbits,r2               ;get size of bitarray
        clrl    r3                      ;clear loop counter
1$:     insv    #0,#0,#1,_a             ;set bit zero to zero
        movl    #1,r4                   ;set pos to 1
2$:     insv    #1,r4,#1,_a             ;set next bit to 1
        aoblss  r2,r4,2$                ;loop through array
        calls   #0,_check               ;check the array
        clrl    r4                      ;clear bits counter
3$:     calls   #0,_rotate              ;rotate the array
        subl3   r4,#nbits-1,r0          ;check the zero bit
        extzv   r0,#1,_a,r0             ;get it. is it zero ?
        beql    4$                      ;if eql yes, hurray !
        jsb     _abort                  ;no, die
4$:     aoblss  r2,r4,3$                ;do a complete loop
        calls   #0,_check               ;check array
        aoblss  #ntimes,r3,1$           ;loop ntimes times
        ret                             ;return to caller

; _rotate

;..first=r4
;..i=r3

        .entry  _rotate,^m<r5,r4,r3,r2>

        extzv   #0,#1,_a,r4             ;get first bit
        movl    #1,r3                   ;clear it
1$:     extzv   r3,#1,_a,r0             ;get it
        decl    r3
        insv    r0,r3,#1,_a             ;store it
        acbl    #nbits,#2,r3,1$         ;loop thru bitarray
        insv    r4,#nbits-1,#1,_a       ;restore first bit at end
        ret                             ;return to caller

; _check

;..i=r4
```

```
        .entry  _check,^m<r5,r4,r3,r2>

        extzv   #0,#1,_a,r0             ;get first bit
        beql    r$                      ;if eql yes, ok
1$:     jsb     _abort                  ;go away
2$:     movl    #1,r4                   ;init counter
3$:     extzv   r4,#1,_a,r0             ;get next bit
        beql    1$                      ;if eql error
        aoblss  #nbits,r4,3$            ;loop thru array
        ret                             ;return to caller

        .end
```

Appendix XIV Benchmark 3 VAX11 assembler programma

```
        .title  b3

        .psect  b3$$data,noexe,page

ntimes=2
_tab:   .long   0,0,1
        .long   0,1,2
        .long   2,0,3
        .long   3,3,61
        .long   3,5,253
        .long   3,7,1021
        .long   -1

        .psect  b3$$code,nowrt,page

; _main

;..i=r4
;..n=r3

        .entry  _main,^m<r5,r4,r3,r2>

        clrl    r3                      ;clear counter
1$:     clrl    r4                      ;clear index
        brb     4$
2$:     pushl   $tab=[r4]               ;push first parameter
        pushl   _tab[r4]                ;push second parameter
        calls   #2,_ack                 ;perform ackermann function
        cmpl    _tab+8[r4],r0           ;correct answer ?
        beql    3$                      ;if eql yes
        jsb     _abort                  ;no, say so
3$:     addl2   #3,r4                   ;update index
4$:     tstl    _tab[r4]                ;end of table ?
        bgeq    2$                      ;if geq no
        aoblss  #ntimes,r3,1$           ;ntimes
        ret                             ;return to caller

; _ack

..m=4
..n=8

        .entry  _ack,^m<r5,r4,r3,r2>

        tstl    ..m(ap)                 ;parameter equal zero ?
        bneq    4$                      ;in neq no
        addl3   ..n(ap),#1,r0           ;yes, add one to 2nd parameter
        brb     3$
1$:     pushl   #1                      ;push 1
2$:     subl3   #1,..m(ap),-(sp)        ;push first param minus 1
        calls   #2,_ack                 ;perform ack function
3$:     ret                             ;return to caller
4$:     tstl    ..n(ap)                 ;2nd param equal 0 ?
        beql    1$                      ;if eql yes
```

```
subl3   #1,..n(ap),-(sp)        ;push 2nd param minus 1
pushl   ..m(ap)                 ;push first param
calls   #2,_ack                 ;perform ack
pushl   r0                      ;push result
brb     2$                      ;

.end
```

Appendix XV Benchmark 4 VAX11 assembler programma

```
        .title  b4

        .psect  b4$$data,noexe,page

ntimes=1000

_x:     .double 0
_y:     .double 0
_x1:    .double 0
_y1:    .double 0
_m10:   .double 0
_m11:   .double 0
_m00:   .double 0.999847695156391
_m01:   .double 0.017452406437284
_eps:   .double 0.000001

        .psect  b4$$code,nowrt,page

; _main

;..i=r4
;..n=r5

        .entry  _main,^m<r5,r4,r3,r2>

        mnegd   _m01,_m10               ;copy values
        movd    _m00,_m11               ;
        clrl    r5                      ;init counter
1$:     movd    #1.0,_x                 ;set up x
        clrd    _y                      ;and y
        clrl    r4                      ;set up index
2$:     muld3   _m00,_x,r0              ;
        muld3   _m01,_y,r2              ;
        addd3   r0,r2,_x1               ;
        muld3   _m10,_x,r0              ;
        muld3   _m11,_y,r2              ;
        addd3   r0,r2,_y1               ;
        movd    _x1,_x                  ;
        movd    _y1,_y                  ;
        aoblss  #360,r4,2$              ;thru 360 degrees ?
        movd    _eps,-(sp)              ;
        movd    #1.0,-(sp)              ;
        movd    _x,-(sp)                ;
        calls   #3,_diff                ;within tolerance ?
        tstl    r0                      ;
        beql    3$                      ;if eql yes
        jsb     _abort                  ;no
3$:     movd    _eps,-(sp)              ;
        clrd    -(sp)                   ;
        movd    _y,-(sp)                ;
        calls   #3,_diff                ;within tolerance ?
        tstl    r0                      ;
        beql    4$                      ;if eql yes
        jsb     _abort                  ;no
```

```
4$:     aoblss  #ntimes,r5,5$           ;intimes
        ret                             ;return to caller
5$:     brw     1$

; _diff

..xa=4
..xb=12
..e=20

        .entry  _diff,^m<r5,r4,r3,r2>

        clrl    r0                      ;set up return value
        subd3   ..xa(ap),..xb(ap),r2    ;difference between params
        bgeq    1$                      ;=> zero ?
        mnegd   r2,r2                   ;make sure positive
1$:     cmpd    r2,..e(ap)              ;compare with tolerance
        bleq    2$                      ;if leq ok
        movl    #1,r0                   ;set error
2$:     ret                             ;return to caller

        .end
```

Appendix XVI Benchmark 1 Interdata 8/32 assembler programma

```
r0      equ     0
r1      equ     1
r2      equ     2
r3      equ     3
r4      equ     4
r5      equ     5
r6      equ     6
r7      equ     7
r8      equ     8
r9      equ     9
r10     equ     10
r11     equ     11
r12     equ     12
r13     equ     13
r14     equ     14
r15     equ     15
sp      equ     7
rf      equ     15
        entry   a
a       equ     *
        ds      504
        entry   rotate
        pure

rotate  equ     *
        sai     sp,l.1
        stm     r11,0(sp)
* first = r13
* s = r12
* t = r11
*@14
        li      r11,a           . =
        lr      r12,r11         . =
*@15
        lb      r13,0(r12)      . =
        ais     r12,1           . ++post
l2      equ     *
*@16
        lb      r0,0(r12)       . load value
        lr      r0,r0
        be      l3
*@17
        lb      r0,0(r12)       . load value
        stb     r0,0(r11)       . =
        ais     r11,1           . ++post
        ais     r12,1           . ++post
        b       l2
l3      equ     *
*@18
        stb     r13,0(r11)      . =
        ais     r11,1           . ++post
l1      equ     *
        lm      r11,0(sp)
        aai     sp,l.1
```

```
        br      rf
1.1     equ     20
        entry   check
        pure

check   equ     *
        sai     sp,1.2
        stm     r12,0(sp)
        ldar    r14,sp
* i = r13
* s = r12
*@25
        li      r12,4           . =
*@27
        lb      r0,0(r12)       . load value
        ais     r12,1           . ++post
        cli     r0,y'78'        . /=
        be      15
*@27
        bal     rf,abort        . call
15      equ     *
16      equ     *
*@28
        lb      r0,0(r12)       . load value
        lr      r0,r0
        be      17
*@30
        lb      r0,0(r12)       . load value
        ais     r12,1           . ++post
        cli     r0,y'61'        . !=
        be      18
*@30
        bal     rf,abort        . call
18      equ     *
        b       16
17      equ     *
14      equ     *
        lm      r12,0(r14)
        aai     sp,1.2
        br      rf
1.2     equ     16
        entry   main
        pure

main    equ     *
        la      sp,stack
* i = r13
* n = r12
*@36
        lis     r12,0           . =
110     equ     *
*@36
        ci      r12,y'28'       . <
        bnl     111
*@37
        li      r0,y'78'        . load value
```

```
        stb     r0,a            . =
*@38
        li      r13,y'1'        . =
113     equ     *
*@38
        ci      r13,y'1f4'      . <
        bnl     114
*@39
        li      r0,y'61'        . load value
        stb     r0,a(r13)       . =
115     equ     *
*@38
        ais     r13,1           . ++post
        b       113
114     equ     *
*@40
        li      r0,y'0'         . load value
        stb     r0,500+a        . =
*@41
        bal     rf,check        . call
*@42
        li      r13,y'0'        . =
116     equ     *
*@42
        ci      r13,y'1f4'      . <
        bnl     117
*@43
        bal     rf,rotate       . call
*@45
        li      r1,y'1f4'       . load value
        sr      r1,r13          . -
        lb      r1,-1+a(r1)     . load value
        lr      r0,r1
        cli     r0,y'78'        . !=
        be      119
*@45
        bal     rf,abort        . call
119     equ     *
118     equ     *
*@42
        ais     r13,1           . ++post
        b       116
117     equ     *
*@47
        bal     rf,check        . call
112     equ     *
*@36
        ais     r12,1           . ++post
        b       110
111     equ     *
19      equ     *
        svc     3,0
abort   svc     2,pa
        svc     3,1
        align   adc
        impur
```

```
        dsf     5
stack   equ     *
pa      db      0,2,0,0
        end
```

Appendix XVII Benchmark 1 Interdata 8/32 hercodering

```
r0      equ     0
r1      equ     1
r2      equ     2
r3      equ     3
r4      equ     4
r5      equ     5
r6      equ     6
r7      equ     7
r8      equ     8
r9      equ     9
r10     equ     10
r11     equ     11
r12     equ     12
r13     equ     13
r14     equ     14
r15     equ     15
sp      equ     7
rf      equ     15
b1opt   equ     *
        lis     r1,0
        lis     r2,1
        lhi     r3,38
l10     lhi     r0,x'78'
        stb     r0,a
        lis     r4,1
        lis     r5,1
        lhi     r6,499
        lhi     r0,x'61'
l13     stb     r0,a(r4)
        bxle    r4,l13
        sr      r0,r0
        stb     r0,500+a
        bal     rf,check
        lis     r4,0
        lis     r5,1
        lhi     r6,499
l14     bal     rf,rotate
        sr      r7,r7
        sr      r7,r4
        lb      r0,a+499(r7)
        chi     r0,x'78'
        bne     abort
        bxle    r4,l14
        bal     rf,check
        bxle    r1,l10
        svc     3,0

* rotate char string in aray a
rotate  equ     *
        la      r11,a
        lr      r12,r11
        lb      r13,0(r12)
        ais     r12,1
l2      lb      r0,0(r12)
```

```
        lr      r0,r0
        be      l3
        stb     r0,0(r11)
        ais     r12,1
        ais     r11,1
        b       l2
l3      stb     r13,0(r11)
        ais     r11,1
        br      rf
*
* check rotate is ok
*


check   equ     *
        la      r12,a
        lb      r0,0(r12)
        ais     r12,1
        clhi    r0,x'78'
        bne     abort
l6      lb      r0,0(r12)
        be      l7
        ais     r12,1
        clhi    r0,x'61'
        bne     abort
        b       l6
l7      br      rf
*
abort   svc     2,pa
        svc     3,1
*
*  data areas
*
        align   4
a       ds      504
pa      db      0,1,0,0
        end     b1opt
```