(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 101819517 B

(45)授权公告日 2013.05.22

(21)申请号 201010185646.3

(22)申请日 2010.05.19

(30)优先权数据

61/179,616 2009.05.19 US

61/228,296 2009.07.24 US

12/572,045 2009.10.01 US

(73)专利权人 威盛电子股份有限公司

地址 中国台湾台北县

(72)发明人 汤玛斯·C·麦当劳 约翰·L·唐肯

(74)专利代理机构 北京市柳沈律师事务所 11105

代理人 钱大勇

(51)Int.Cl.

G06F 9/30(2006.01)

G06F 9/38(2006.01)

(56)对比文件

CN 101377735 A,2009.03.04, 全文.

CN 101256504 A,2008.09.03, 全文.

EP 1028370 B1,2004.09.15, 全文.

US 2006/0206690 A1,2006.09.14, 全文.

审查员 杨庆丽

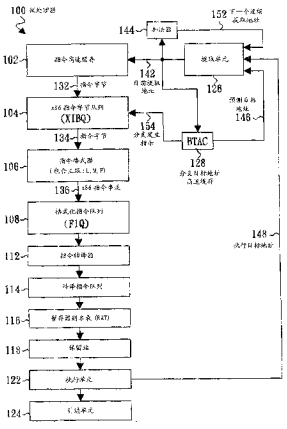
权利要求书2页 说明书20页 附图20页

(54)发明名称

适用于微处理器的装置及方法

(57)摘要

一种适用于微处理器的装置和方法，其中该装置用以有效地使用微处理器以决定一指令字节串流中的一指令的一指令长度，该微处理器具可变指令长度的指令集架构，该装置包含：多个组合逻辑单元，分别对应于该指令字节串流的每一指令字节，每一该组合逻辑单元接收相应的该指令字节及其下一指令字节，用以产生一第一长度、一第二长度及一选择控制信号；一多工器，对应于每一该组合逻辑单元，根据该选择控制信号以选择及输出下列的输入之一：0值、接收自相应于该指令字节的后续三指令字节的该组合逻辑单元的该第二长度；及一加法器，对应于每一该组合逻辑单元及该多工器，用以加总该第一长度和该多工器的输出，用以产生该指令长度。



CN 101819517 B

1. 一种适用于微处理器的装置，用以有效地使用微处理器以决定一指令字节串流中的一指令的一指令长度，该微处理器具可变指令长度的指令集架构，该装置包含：

多个组合逻辑单元，分别对应于该指令字节串流的一个指令字节，每一该组合逻辑单元接收相应的该指令字节及该指令字节的后续下一个指令字节，用以产生一第一长度值、一第二长度值及一选择控制信号；

一多工器，对应于每一该组合逻辑单元，根据该选择控制信号以选择及输出下列的输入之一：0 值、接收自相应于该指令字节的后续三个指令字节对应的组合逻辑单元产生的三个第二长度值；及

一加法器，对应于每一该组合逻辑单元及该多工器，用以加总该第一长度值和该多工器的输出，用以产生该指令长度。

2. 根据权利要求 1 所述的装置，其中当相应的该指令字节为该指令的第一非前置字节时，则上述加法器输出的该指令长度为真正指令长度。

3. 根据权利要求 1 所述的装置，其中上述的指令集架构为 x86 指令集架构，其中上述的组合逻辑单元还接收一地址 / 操作数尺寸消息，并根据该地址 / 操作数尺寸消息、相应的该指令字节及下一指令字节，以产生该第一长度值、该第二长度值和该选择控制信号。

4. 根据权利要求 3 所述的装置，其中上述的组合逻辑单元假设相应的该指令字节及其下一指令字节为该指令的前二操作码字节，用以产生该第一长度值，其中该第一长度值为下列之和：该指令的操作码字节的数目和该指令的立即数据的尺寸，第一长度值是由该指令的前二操作码字节所决定。

5. 根据权利要求 3 所述的装置，其中上述的组合逻辑单元假设相应的该指令字节为该指令的 x86 ModR/M 字节，用以产生该第二长度值，其中该第二长度值为下列之和：1 值、x86 SIB 字节数目和指令位移的尺寸。

6. 根据权利要求 3 所述的装置，其中上述的组合逻辑单元根据下列指令形式之一以产生该选择控制信号以选择并输出如下输入之一：

当该指令的形式仅为操作码，则选择 0 值输入；

当该指令的形式为 0x0F+ 操作码，则选择 0 值输入；

当该指令的形式为操作码 +ModR/M 字节，则选择接收自相应于下一指令字节的该组合逻辑单元的该第二长度值；

当该指令的形式为 0x0F+ 操作码 +ModR/M 字节，则选择接收自相应于下二指令字节的该组合逻辑单元的该第二长度值；及

当该指令的形式为 0x0F+0x38/0x3A+ 操作码 +ModR/M 字节，则选择接收自相应于下三指令字节的该组合逻辑单元的该第二长度值。

7. 根据权利要求 3 所述的装置，其中上述的地址 / 操作数尺寸消息是于该加法器产生该指令长度的时钟周期的前一周期内，根据一指令长度修改前置字节的解码所产生的。

8. 一种适用于微处理器的方法，用以有效地使用微处理器以决定一指令字节串流中的一指令的一指令长度，该微处理器具可变指令长度的指令集架构，该方法包含：

接收每一指令字节及该指令字节的后续下一指令字节，据以产生一第一长度值、一第二长度值及一选择控制信号；

对于每一指令字节，根据该选择控制信号以选择及输出下列的输入之一：0 值、相应于

该指令字节的后续三个指令字节产生的三个第二长度值；及

加总该第一长度值和该选择的输出，用以产生该指令长度。

9. 根据权利要求 8 所述的方法，其中当相应的该指令字节为该指令的第一非前置字节时，则上述加总所产生的该指令长度为真正指令长度。

10. 根据权利要求 8 所述的方法，其中上述的指令集架构为 x86 指令集架构，其中上述产生该第一长度值、该第二长度值和该选择控制信号的步骤的执行是根据一地址 / 操作数尺寸消息，其连同相应的该指令字节一起被接收。

11. 根据权利要求 10 所述的方法，其中上述产生该第一长度值的步骤是假设相应的该指令字节及其下一指令字节为该指令的前二操作码字节，其中该第一长度值为下列之和：该指令的操作码字节的数目和该指令的立即数据的尺寸，第一长度值是由该指令的前二操作码字节所决定。

12. 根据权利要求 10 所述的方法，其中上述产生该第二长度值的步骤是假设相应的该指令字节为该指令的 x86ModR/M 字节，其中该第二长度值为下列之和：1 值、x86SIB 字节数目和指令位移的尺寸。

13. 根据权利要求 10 所述的方法，其中上述产生该选择控制信号以选择及输出如下输入之一的步骤是根据下列指令形式之一以执行：

当该指令的形式仅为操作码，则选择 0 值输入；

当该指令的形式为 0x0F+ 操作码，则选择 0 值输入；

当该指令的形式为操作码 +ModR/M 字节，则选择相应于下一指令字节的该第二长度值；

当该指令的形式为 0x0F+ 操作码 +ModR/M 字节，则选择相应于下二指令字节的该第二长度值；及

当该指令的形式为 0x0F+0x38/0x3A+ 操作码 +ModR/M 字节，则选择相应于下三指令字节的该第二长度值。

14. 根据权利要求 10 所述的方法，其中上述的地址 / 操作数尺寸消息是于该加总步骤的时钟周期的前一周期内，根据一指令长度修改前置字节的解码所产生的。

适用于微处理器的装置及方法

技术领域

[0001] 本发明是有关微处理器领域，特别是关于从一种具有可变长度指令的指令集架构的微处理器的指令字节串流中取得指令。

背景技术

[0002] 微处理器包含一或多个执行单元，用以进行实际的指令执行。超纯量(superscalar)微处理器可于每一时钟周期内发出多个指令至各个执行单元，因而得以增进总处理能力或增进每一时钟周期内的平均指令。然而，微处理器管线上端的指令提取及解码功能必须以有效率的速度来提供一指令串流给执行单元，藉以有效地使用执行单元及增进总处理能力。x86 架构由于其指令长度并非固定，因此使得此工作更加困难，在此架构下，其每一指令的长度是变动的，此将于以下详述。因此，x86 微处理器必须包含很多的逻辑电路以处理进来的指令字节串流，以决定指令的开始及结束位置。因此，必须增进 x86 微处理器解析指令字节串流以得到各个指令的处理速率。

发明内容

[0003] 根据本发明特征之一，本发明提供一种适用于微处理器的装置，用以有效地使用微处理器以决定一指令字节串流中的一指令的一指令长度，该微处理器具可变指令长度的指令集架构，该装置包含：多个组合逻辑单元，分别对应于该指令字节串流的每一指令字节，每一该组合逻辑单元接收相应的该指令字节及其下一指令字节，用以产生一第一长度、一第二长度及一选择控制信号；一多工器，对应于每一该组合逻辑单元，根据该选择控制信号以选择及输出下列的输入之一：0 值、接收自相应于该指令字节的后续三指令字节的该组合逻辑单元的该第二长度；及一加法器，对应于每一该组合逻辑单元及该多工器，用以加总该第一长度和该多工器的输出，用以产生该指令长度。

[0004] 根据本发明特征之一，本发明提供一种适用于微处理器的方法，用以有效地使用微处理器以决定一指令字节串流中的一指令的一指令长度，该微处理器具可变指令长度的指令集架构，该方法包含：接收每一指令字节及其下一指令字节，据以产生一第一长度、一第二长度及一选择控制信号；对于每一指令字节，根据该选择控制信号以选择及输出下列的输入之一：0 值、相应于该指令字节的后续三指令字节的该第二长度；及加总该第一长度和该选择的输出，用以产生该指令长度。

附图说明

[0005] 图 1 显示本发明实施例的微处理器的方块图。

[0006] 图 2 显示图 1 的指令格式器的 L 级的方块图。

[0007] 图 3 显示图 2 的累积前置消息 238。

[0008] 图 4 显示图 1 的微处理器的操作。

[0009] 图 5 显示图 1 的指令格式器的部分 L 级和 M 级方块图。

[0010] 图 6 显示图 5 所示的微处理器元件的操作流程图，用以自指令字节串流中取出指令（在一实施例中最多可取出三指令），其不会产生时间延迟且与指令中的前置字节数目无关。

[0011] 图 7 显示图 1 的指令格式器的一部分的方块图。

[0012] 图 8a 和图 8b 显示图 7 的部分指令格式器的操作流程图。

[0013] 图 9 显示图 5 的多工队列的详细方块图。

[0014] 图 10 显示图 1 的指令格式器的部分 M 级的方块图。

[0015] 图 11 显示图 5 的 M 级控制逻辑单元的方块图。

[0016] 图 12 显示图 1 的指令格式器的部分 M 级的操作流程图。

[0017] 图 13 显示图 5 的多工队列于连续两个时钟周期的内容，以例示 M 级的操作。

[0018] 图 14 显示图 5 的多工队列于连续两个时钟周期的内容，以例示 M 级的操作。

[0019] 图 15 显示图 14 中指令格式器于一时钟周期内，将含有最多四十个指令字节的三个指令取得并传送出去。

[0020] 图 16 显示图 1 的 BTAC 作了不良预测因而造成微处理器的分支错误，亦即，图 1 的分支发生指示为逻辑真值但非为指令的操作码。

[0021] 图 17 显示涟波逻辑单元输出的组成信号。

[0022] 图 18 显示图 1 的微处理器的操作流程图。

[0023] 图 19 显示图 2 的长度解码器的详细方块图。

[0024] 图 20 显示十六个长度解码器的配置。

[0025] 图 21 显示图 20 的长度解码器的操作流程图。

[0026] [主要元件标号说明]

[0027]	100	微处理器	102	指令高速缓存
[0028]	104	x86 指令字节队列	106	指令格式器
[0029]	108	格式化指令队列	112	指令转译器
[0030]	114	转译指令队列	116	暂存器别名表
[0031]	118	保留站	122	执行单元
[0032]	124	引退单元	126	提取单元
[0033]	128	分支目标地址高速缓存	132	指令字节
[0034]	134	指令字节	136	x86 指令串流
[0035]	142	目前提取地址	144	加法器
[0036]	146	预测目标地址	148	执行目标地址
[0037]	152	下一个连续提取地址	154	分支发生指示
[0038]	202	长度解码器	204	涟波逻辑单元
[0039]	208	控制逻辑单元	212	长度解码器的输出
[0040]	214	涟波逻辑单元的输出	218	操作数及地址尺寸
[0041]	222	指令长度	224	解码任一前置指示符

[0042] 226 解码 LMP 指示符 228 受 LMP 影响指示符

[0043] 229 前置消息 232 开始位

[0044] 234 结束位 236 有效位

[0045] 238 累积前置消息 252 预设操作数及地址尺寸

[0046] 302 OS 304 AS

[0047] 306 REX 出现 308 REX.W

[0048] 312 REX.R 314 REX.X

[0049] 316 REX.B 318 REP

[0050] 322 REPNE 324 LOCK

[0051] 326 片段超出出现 328 编码段超出 [2:0]

[0052] 332 任一前置出现 402-414 步骤

[0053] 502 多工队列 504 I1 多工器

[0054] 506 I2 多工器 508 I3 多工器

[0055] 512 M 级控制逻辑单元 514 控制信号

[0056] 516 控制信号 518 控制信号

[0057] 524 第一指令 I1 526 第二指令 I2

[0058] 528 第三指令 I3 534,536,538 有效指示符

[0059] 602-608 步骤 702 XIBQ 控制逻辑单元

[0060] 802-824 步骤 1002 累积前置阵列

[0061] 1004 指令字节阵列 1102 减法器

[0062] 1104 部分 LEN 1106 剩余 LEN1

[0063] 1108 字节位置 END1 1112 字节位置 END0

[0064] 1114 多工器 1116 加法器

[0065] 1118 暂存器 1122 指令长度 LEN1

[0066] 1201-1222 步骤 1702 不良 BTAC 位

[0067] 1802-1816 步骤 1902 可编程逻辑阵列 (PLA)

[0068] 1904 加法器 1906 多工器

[0069] 1912 eaLen 值 1914 控制信号

[0070] 1916 immLen 值 1918 eaLen 值

[0071] 2102-2116 步骤

具体实施方式

[0072] 图 1 显示本发明实施例的微处理器 100 的方块图。微处理器 100 包含由多

级或多个功能单元所组成的管线 (pipeline)，其包含四级指令高速缓存 (four-stage instruction cache)102、x86指令字节队列(x86 instructionbyte queue，XIBQ)104、指令格式器 (instruction formatter)106(其包含三级L、M及F)、格式化指令队列 (formatted instruction queue)108、指令转译器 (instruction translator)112、转译指令队列 (translatedinstruction queue)114、暂存器别名表 (register alias table)116、保留站 (reservation station)118、执行单元 (execution units)122及引退单元 (retire unit)124。微处理器100还包含提取单元 (fetch unit)126，其提供目前提取地址142给指令高速缓存102，用以选择一指令字节 (byte)132快取列至XIBQ104。微处理器100还包含加法器144，其增加目前提取地址142以产生下一个连续提取地址152，再反馈至提取单元126。提取单元126还从分支目标地址高速缓存 (branch target address cache，BTAC)128接收预测目标地址146。最后，提取单元126从执行单元122接收执行目标地址 (executedtarget address)148。

[0073] XIBQ104的队列含有多个项目 (entry)，每一个项目包含来自指令高速缓存102的十六字节数据。再者，XIBQ104的每一个项目包含数据字节相关的预解码 (pre-decoded) 消息。预解码消息为当数据字节从指令高速缓存102流至XIBQ104时所产生的。来自XIBQ104的快取数据为指令字节134串流，其形式为多个十六字节区块，然而并不知道串流内或区块内的x86指令的开始或结束位置。指令格式器106即用以决定串流内每一指令的开始及结束字节，从而将字节串流分离为x86指令串流136，其再馈至并储存于格式化指令队列108，以待微处理器100管线的其它部分进行处理。当发生重置或执行/预测到流量控制指令(例如跳越 (jump) 指令、子例程呼叫 (subroutine call) 指令或自子例程返回指令)时，则提供重置地址或分支目标地址给指令格式器106作为指令指针 (pointer)，用以致能指令格式器106，使其决定出指令串流的目前的十六字节区块内的第一有效指令的第一字节。因此，指令格式器106即可根据第一目标指令的开始位置加上第一目标指令的长度，以决定下一指令的开始位置。指令格式器106重复上述程序，直到执行或预测到另一流量控制指令。

[0074] BTAC128还提供分支发生 (taken) 指示154给XIBQ104。指令高速缓存102提供给XIBQ104的每一指令字节132对应有一个分支发生指示154。分支发生指示154用以表示BTAC128预测提供给XIBQ104的指令字节132列是否具有分支指令；如果为是，则提取单元126将会选取BTAC128所提供的预测目标地址146。详而言之，BTAC128对于分支指令的第一字节(即使该第一字节为前置字节)会相应输出逻辑真值的分支发生指示154，但对于指令的其它字节则会输出逻辑假值的分支发生指示154。

[0075] 微处理器100为x86架构的微处理器100。当微处理器可正确地执行专为x86微处理器所执行的大部分应用程序时，则该微处理器即可视为x86架构的微处理器。当可以得到预期结果时，则该应用程序即可视为可正确地执行。X86架构的特征之一为其指令集架构中的指令长度是可变的，而非像一些指令集架构中的指令长度是固定的。再者，对于某一x86操作码 (opcode)，可能会因为操作码之前是否具有前置 (prefix) 而影响指令的长度。此外，一些指令的长度可能为微处理器100操作模式下的预设操作数 (operand) 及/或地址尺寸的函数(例如码段描述符 (code segment descriptor) 的D位，或者微处理器100是否操作于IA-32e或64位模式)。最后，于预设地址/操作数尺寸之外，指令还可

包含一长度修改前置 (length-modifying prefix),用以选择地址 / 操作数尺寸。例如,可使用操作数尺寸 (operand size,OS) 前置 (0x66)、地址尺寸 (AS) 前置 (0x67) 及 REX 前置 (0x4x) 的 REX.W 位 (位 3) 以改变预设的地址 / 操作数尺寸。英特尔 (Intel) 公司称这些为长度改变前置 (length-changing prefix,LCP),然而在本说明书中则称为长度修改前置 (length-modifying prefix,LMP)。X86 指令的格式及长度为大家所熟知,细节可参考 IA-32 英特尔架构软件开发手册 (IA-32Intel ArchitectureSoftware Developer's Manual),第 2A 集的第二章 :指令集参考 (InstructionSet Reference),A-M,公元 2006 年六月。

[0076] 根据英特尔 64 及 IA-32 架构最佳化参考手册 (Intel® 64 and IA-32Architectures Optimization Reference Manua l),公元 2009 年三月,页 3-21 至 3-23(可自下列网页下载 http://www.intel.com/Assets/PDF/manual/248966.pdf) :「当预解码器于提取列中遇到 LCP,则必须使用较慢的长度解码算法。当使用较慢的长度解码算法时,预解码器于六个周期内进行解码,而非一般的一个周期。于机器管线内的队列 (queuing) 一般是无法避免 LCP 造成的延迟。」

[0077] 图 2 显示图 1 的指令格式器 106 的 L 级的方块图。指令格式器 106 包含多个长度解码器 202,其输出 212 分别耦接至多个涟波 (ripple) 逻辑单元 204,涟波逻辑单元 204 的输出 214 耦接至控制逻辑单元 208 并提供给指令格式器 106 的 M 级。在一实施例中,长度解码器 202 于微处理器 100 的二相位时钟信号的第一相位期间产生输出 212,而涟波逻辑单元 204 于二相位时钟信号的第二相位期间产生输出 214。

[0078] 长度解码器 202 从 XIBQ104 接收指令字节 134。在一实施例中,XIBQ104 的每一项目宽度为十六字节,因而相应有十六个长度解码器 202,如图 2 所示的 0 至 15。每一个长度解码器 202 自 XIBQ104 的底部项目接收并解码相应的指令字节。此外,每一个长度解码器 202 接收并解码接下来的三个相邻指令字节。对于最后三个长度解码器 202,其自 XIBQ104 的底部倒数第二个项目接收一或多个指令字节 (如果 XIBQ104 的底部倒数第二个项目为无效,则最后三个长度解码器 202 必须等待下一个时钟周期中产生有效输出)。长度解码器 202 的细节将于图 19 说明。藉此,使得长度解码器 202 可以决定及输出 XIBQ104 的底部项目中的指令的指令长度 222。在一实施例中,指令长度 222 表示该指令除了前置字节以外的字节数目。换句话说,指令长度 222 表示指令当中,从操作码至最后一个字节的字节数目。具体来说,由对应于指令的第一指令字节的长度解码器 202 所输出的指令长度为指令长度 222。

[0079] 为了产生指令长度 222,长度解码器 202 还使用接收自控制逻辑单元 208 的操作数及地址尺寸 218。控制逻辑单元 208 对于每一指令字节 134 会输出操作数及地址尺寸 218。控制逻辑单元 208 根据目前微处理器 100 的预设操作数及地址尺寸 252 和涟波逻辑单元 204 的输出 214 以决定操作数及地址尺寸 218。如果涟波逻辑单元 204 的输出 214 表示指令中无 LMP,则控制逻辑单元 208 对于每一指令字节会输出预设操作数及地址尺寸给相应的长度解码器 202。然而,如果涟波逻辑单元 204 的输出 214 表示指令中有一或多个 LMP,则控制逻辑单元 208 对于每一指令字节会修改预设操作数及地址尺寸 252 而输出操作数及地址尺寸 218 给相应的长度解码器 202,其中控制逻辑单元 208 根据 OS 302、AS 304 及 REX.W 308 位的值修改预设操作数及地址尺寸 252,这些位包含于涟波逻辑单元 204 的输出 214 的累积前置消息 238 中,如图 3 所示。

[0080] 如图 2 所示，每一长度解码器 202 的输出 212 包含指令字节 134、指令长度 222、解码任一前置指示符 (decoded any prefix indicator)224、解码 LMP 指示符 (decoded LMP indicator)226、受 LMP 影响指示符 (susceptible toLMP indicator)228 及前置消息 229。

[0081] 当长度解码器 202 所解码的字节对应到任一 x86 前置（无论其是否为 LMP)，则解码任一前置指示符 224 为逻辑真值；否则，为逻辑假值。

[0082] 当长度解码器 202 所解码的字节对应到任一 x86LMP，亦即 OS 前置 (0x66)、AS 前置 (0x67) 或 REX.W 前置 (0x48-0x4F)，则解码 LMP 指示符 226 为逻辑真值；否则，为逻辑假值。

[0083] 当长度解码器 202 所解码的字节是操作码字节，其中操作码的指令长度不受 LMP 影响（例如，OS 前置对于一些 SIMD 指令为强制的，因此不能改变其长度），则受 LMP 影响指示符 228 为逻辑假值；否则，为逻辑真值。

[0084] 前置消息 229 包含多个位 (bit)，用以表示指令字节是否具有各种 x86 前置其中之一。这些位类似于图 3 所示的累积前置消息 238。然而，长度解码器 202 输出的前置消息 229 仅表示单一前置，亦即，受长度解码器 202 解码的单一对应的指令字节的前置值。相反的，由于涟波逻辑单元 204 将所有长度解码器 202 提供的前置消息 229 予以累积，因此累积前置消息 238 则表示指令中的所有前置。

[0085] 如图 2 所示，每一涟波逻辑单元 204 的输出 214 包含指令字节 134、开始位 232、结束位 234、有效位 236 及累积前置消息 238。每一涟波逻辑单元 204 的输出 214 还馈至下一相邻的涟波逻辑单元 204。在一实施例中，十六个涟波逻辑单元 204 组织成四个逻辑区块，每一区块处理四个指令字节及其相关消息。每一涟波逻辑单元区块 204 还输出相应的指令字节。

[0086] 当涟波逻辑单元 204 所处理的字节为指令的操作码字节时（例如指令的第一字节非为前置字节），则开始位 232 为逻辑真值。指令格式器 106 增加一指标，其指向所有前置字节，使得当指针指向一非前置字节时，该指针将会指向指令的操作数字节。

[0087] 当涟波逻辑单元 204 所处理的字节为指令的最后字节时，则结束位 234 为逻辑真值；否则，为逻辑假值。

[0088] 从涟波逻辑单元 204 输出的十六个有效位 236 的第一个开始，直到出现第一个未处理的 LMP 为止，每一有效位 236 为逻辑真值。

[0089] 累积前置消息 238 显示于图 3 并讨论如上。控制逻辑单元 208 使用累积前置消息 238 并配合有效位 236，以决定是否使用预设操作数及地址尺寸 252 或对其进行修改。

[0090] 值得注意的是，长度解码器 202 的输出 212 属于一种试验性质。换句话说，其产生输出时并不知道相关指令字节在指令内的地址。尤其是，与前置相关的指示符 224/226/228/229 是在假设该字节为有效前置的前提下所产生的，而此假设可能是一个错误的假设。因此，该字节可能恰巧具有一前置的值，但该字节其实是具有与 LMP 相同的值的位移 (displacement) 字节。例如，0x67 为 AS 前置的值，其为 LMP。然而，地址位移字节或立即数据值 (immediatedata value) 字节或 Mod R/M 字节或指令的 SIB 字节皆不是前置字节，但可能具有 0x67 值。仅当指令字节的目前区块中的所有 LMP 都已处理，才能确定相应于区块中所有字节的输出 212 及 214 都为正确。

[0091] 如果在目前时钟周期内，XIBQ104 项目中的所有指令字节并未被解码出任何 LMP，

则L级会在单一时钟周期内产生整个项目的涟波逻辑单元204输出214(特别是开始位232和结束位234)。如果XIBQ104目前的项目中被解码出一或多个LMP,则产生具正确开始位232和结束位234的涟波逻辑单元204输出214所需的时钟周期数为N+1,其中N为XIBQ104目前的项目中具有至少一LMP的指令的数目。无论项目中的任一指令的前置数目为多少,L级均可执行上述工作,此显示于图4的流程图中。控制逻辑单元208包含一状态,用以表示指令字节的目前区块中的哪些字节已被处理过,哪些尚未处理。该状态使得控制逻辑单元208可针对每一指令字节产生有效位236及操作数及地址尺寸218。由于具有含LMP的指令的指令字节区块的处理具有迭代(iterative)特性,即使于第一时钟周期时,含LMP的第一指令的指令长度222、开始位232和结束位234可能并不正确;然而,于下一时钟周期时,第一指令和任一不含LMP的相邻指令的指令长度222、开始位232和结束位234则会变为正确;且于接续时钟周期中,第一指令的下一含LMP的指令及其相邻不含LMP的指令的指令长度222、开始位232和结束位234均会正确。在一实施例中,该状态包含十六位暂存器,用以表示相关指令字节是否已被处理。

[0092] [针对含有LMP的指令标示出开始及结束字节]

[0093] 图4显示图1的微处理器100的操作,该流程始于步骤402。

[0094] 于步骤402,控制逻辑单元208输出预设操作数及地址尺寸218给长度解码器202。接着,流程进入步骤404。

[0095] 于步骤404,于时钟周期的第一相位,长度解码器202根据控制逻辑单元208提供的操作数及地址尺寸218,以解码XIBQ104的底部项目的指令字节并产生其输出212。如前所述,对于XIBQ104的底部项目的每一指令字节,长度解码器202的输出212包含指令长度222及与前置相关的指示符224/226/228/229(图2)。接着,流程进入步骤406。

[0096] 于步骤406,于时钟周期的第二相位,涟波逻辑单元204根据长度解码器202的输出212以产生输出214。如前所述,涟波逻辑单元204的输出214包含开始位232、结束位234、有效位236及累积前置消息238(图3)。接着,流程进入步骤408。

[0097] 于步骤408,控制逻辑单元208检视(examine)涟波逻辑单元204的输出214,以判断XIBQ104的底部项目中是否还有任何指令包含未处理的LMP(长度修改前置符)。如果为是,则进入步骤412;否则,进入步骤414。

[0098] 于步骤412,控制逻辑单元208根据涟波逻辑单元204提供的累积前置消息238,以更新内部状态及操作数及地址尺寸。接着,流程返回步骤404,依据新的操作数尺寸及地址尺寸,再次处理底部项目的指令字节。

[0099] 于步骤414,控制逻辑单元208判断底部项目的指令字节已完全处理完,因而将其自XIBQ104移出,并将其连同每一指令字节134相应的涟波逻辑单元204的输出214一起送至M级。特别的是,如前所述,由于涟波逻辑单元204的输出214包含开始位232及结束位234,其表示出指令高速缓存102所提供的指令串流当中每一指令的边界,因而使得指令格式器106的M级和F级得以进一步处理指令串流,并将个别指令置入FIQ(格式化指令队列)108,让指令转译器112进行处理。流程结束于步骤414。

[0100] 根据前述,如果指令字节中未含有LMP(长度修改前置符),则L级可于单一时钟周期中针对XIBQ(x86字节字队列)104的整个项目以产生开始位232及结束位234;如果XIBQ104的项目中有一个或更多指令具有LMP(长度修改前置符),则产生开始位232及结

束位 234 所需的时钟周期数变为 N+1，其中 N 为 XIBQ104 项目中含有至少一 LMP（长度修改前置符）的指令数目，且无论指令中含有的前置数目为何，L 级都可以进行。

[0101] ［累积前置以有效地处理含多个前置字节的指令］

[0102] x86 架构允许指令含有 0 至 14 个前置字节。此造成管线 (pipeline) 前端于处理指令字节串流时的困难。以往于处理含相当数目的前置字节的指令时，会遭遇时间的延迟。根据英特尔 64 及 IA-32 架构最佳化参考手册（Intel® 64and IA-32 Architectures Optimization Reference Manual)，公元 2009 年三月，页 12-5，英特尔针对 ATOM 微架构提到：「含三个以上前置的指令会产生 MSROM 转移，造成前端的二个时钟周期延迟。」根据另一研究文献 - 英特尔及 AMD 中央处理器的微架构 (The microarchitecture of Intel and AMD CPU’s)，作者 Agner Fog，Copenhagen University College of Enginerring，公元 2009 年 5 月 5 日最后一次更新，页 93（可于以下网页下载 www.agner.org/optimize/microarchitecture.pdf)，其提到：「含多个前置的指令需要额外时间以进行解码。P4 的指令解码器于一时钟周期内仅可处理一前置。于 P4 上，含多个前置的指令其每一个前置需花费一时钟周期解码」，且「P4E 的指令解码器可于一时钟周期处理二个前置。因此，单一时钟周期内可解码含有至多二前置的指令，而含三或四个前置的指令则需于二时钟周期内进行解码。P4E 的所以增加此功能，乃因为在 64 位模式下，很多指令都含有二前置（例如操作数尺寸前置及 REX 前置）。」

[0103] 然而，本发明实施例在不需增加时间延迟的条件下，可处理一个指令中架构所允许的所有（至多 14 个）前置字节，无论前置字节的数量为何（只要该前置非为 LMP（长度修改前置符），若该前置为 LMP，则含一或多个前置的每一指令会额外增加一时钟周期的处理时间，如前所述）。本发明实施例的所以能够达成此目的，是因为长度解码器 202 产生前置消息 229，而涟波逻辑单元 204 则累积前置消息 229 以产生累积前置消息 238 给指令的操作码字节，此将于以下详述。

[0104] 图 5 显示图 1 的指令格式器 106 的部分 L 级和 M 级（多工级）方块图。M 级包含多工队列 (mux queue)502。在一实施例中，多工队列 502 包含四个项目，每一项目储存十六字节。多工队列 502 的下一空白项目接收相应涟波逻辑单元 204 的输出 214（图 2)，其包含指令字节 134、开始位 232、结束位 234 及累积前置消息 238。

[0105] M 级还包含 M 级控制逻辑单元 512，其自多工队列 502 的底部项目接收开始 / 结束位 232/234，且（在一实施例中）接收多工队列 502 的底部倒数第二项目 (next-to-bottom entry，NTBE) 的前十字节。根据开始 / 结束位 232/234，M 级控制逻辑单元 512 控制三组多工逻辑单元，分别为 I1 多工器 504、I2 多工器 506 及 I3 多工器 508。I1 多工器 504 输出第一指令 I1524 至指令格式器 106 的 F 级；I2 多工器 506 输出第二指令 I2526 至 F 级；I3 多工器 508 输出第三指令 I3528 至 F 级。此外，M 级控制逻辑单元 512 输出三个有效指示符 534/536/538，用以表示相应的第一、第二、第三指令 524/526/528 是否有效。藉此，M 级得以从指令串流中最多取出 (extract) 三个格式化指令，并在单一时钟周期内将其提供给 F 级。在其它实施例中，M 级可在单一时钟周期内取出并提供多于三个格式化指令给 F 级。三个指令 524/526/528 中的每一指令包含相应指令字节 134，并且其前置字节被置换为相应的累积前置消息 238。换句话说，每一指令 524/526/528 包含操作码字节及指令字节的其它部分以及累积前置消息 238。每一多工器 504/506/508 自多工队列 502 的相应底部项目

分别接收消息 214（但开始位 232、结束位 234 除外），且（在一实施例中）自多工队列 502 的相应 NTBE 接收前十字节，用以个别选取及输出指令 524/526/528。

[0106] 图 6 显示图 5 所示的微处理器 100 元件的操作流程图，用以自指令字节串流中取出指令（在一实施例中最多可取出三指令），其不会产生时间延迟且与指令中的前置字节数目无关。如前所述，涟波逻辑单元 204 会累积前置消息 229 以产生累积前置消息 238 给指令的操作码字节。所示流程始于步骤 602。

[0107] 于步骤 602，于时钟周期的第一相位，长度解码器 202 解码指令字节 134 串流以产生输出 212（图 2），特别是前置消息 229，此和步骤 404 的操作类似。接着，进入步骤 604。

[0108] 于步骤 604，于时钟周期的第二相位，涟波逻辑单元 204 依据前置消息 229 以决定串流的每一个指令中哪一字节为操作码字节（亦即第一非前置字节）。再者，涟波逻辑单元 204 针对指令中的所有（最多为十四个）前置字节累积其前置消息 229，以产生累积前置消息 238 给指令的操作码字节。特别的是，涟波逻辑单元 204 自指令的第一前置字节开始累积前置消息 229，并且逐一累积每个字节的前置消息 229，直到其检测到操作码字节为止。届时，涟波逻辑单元 204 停止前置消息的累积，使得目前指令的累积前置消息 238 不会继续累积到下一指令去。涟波逻辑单元 204 自下一指令的第一前置字节开始进行前置消息 229 的累积，并停止于操作码字节。对于串流中的每一指令，重复此程序。涟波逻辑单元 204 使用长度解码器 202 的另一输出 212 以完成前置消息的累积。例如，如前所述，涟波逻辑单元 204 使用指令长度 222 以决定每一指令的第一字节，其可能为前置字节，用以开始前置消息的累积程序。涟波逻辑单元 204 还使用其它消息 224/226/228 以决定操作码字节的位置，其为不含前置的指令的第一字节（由开始位 232 表示），并决定指令最后字节的位置（由结束位 234 表示）。接着，流程进入步骤 606。

[0109] 于步骤 606，指令字节 134 及相应的开始 / 结束位 232/234、累积前置消息 238 被加载多工队列 502 的下一可用项目中。在一实施例中，步骤 602、604、606 所示的步骤被于单一时钟周期内执行（假设指令不含有 LMP（长度修改前置符））。接着，进入步骤 608。

[0110] 于步骤 608，在下一时钟周期，M 级控制逻辑单元 512 控制多工器 504/506/508，使其至多可取出三个指令。换句话说，不管前置字节的数量为何，M 级不需增加时间延迟而能够取得指令。经多工 (muxed) 后，指令 524/526/528 可各个馈至 F 级。特别的是，M 级随着累积前置消息 238 可取出每一指令的操作码字节及后续字节。F 级依据指令型态、一些可能的例外情形、可配对性 (pairability) 及其它特性以解码指令 524/526/528，以开始指令 524/526/528 的转译。F 级和指令转译器 112 可利用累积前置消息 238。流程结束于步骤 608。

[0111] 本实施例不同于传统的设计。如前所述，涟波逻辑单元 204 较传统来得复杂，其所产生的开始位 232 是指向指令的操作码字节，而非如传统般指向指令的第一字节（其可能为前置字节），且产生累积前置消息 238，因此，无论前置字节的数量为何均可取得指令且不会造成时间延迟（除非是 LMP（长度修改前置），已如前述）。相反地，传统的作法是指出指令实际的第一字节为第一字节，如果指令含有前置字节，则该前置字节被表示为第一指令。当指令含有多个前置字节时，为了除去前置字节，传统的多工逻辑因此会造成时间延迟。

[0112] [当指令部分出现时，以开始 / 结束标示使得快取数据能尽快释放]

[0113] 图 7 显示图 1 的指令格式器 106 的一部分的方块图。在图 1 中，指令高速缓存 102 提供指令字节 132 至 XIBQ104。在一实施例中，指令格式器 106 包含预解码 (pre-decode) 逻辑单元（未显示于图式中），用以对来自指令高速缓存 102 的指令字节 132 进行预解码，而经预解码消息则连同指令字节 132 一并载至 XIBQ104。指令格式器 106 包含 XIBQ 控制逻辑单元 702，其控制 XIBQ104 的项目加载及移出。

[0114] 长度解码器 202 及涟波逻辑单元 204（图 2）自 XIBQ104 接收指令字节 134 并产生输出 214，用以提供给图 5 的多工队列 502 及指令格式器 106 的 M 级控制逻辑单元 512。M 级控制逻辑单元 512 控制多工队列 502 的项目加载及移出。多工队列 502 自其项目中提供消息 214 给多工器 504/506/508 和 M 级控制逻辑单元 512，M 级控制逻辑单元 512 又控制多工器 504/506/508，如前所述。

[0115] 当以下情形时会产生问题：(1)XIBQ104 的底部项目包含有效指令字节但是 NTBE 则未包含；(2) 只有部分的指令（例如指令的第一或第二字节）在底部项目；(3) 部分的指令未提供足够消息让长度解码器 202/ 涟波逻辑单元 204 决定指令长度 222（及开始 / 结束位 232/234），亦即，指令还有一些字节位于 NTBE。例如，假设在 XIBQ104 底部项目的字节 15（亦即最后字节）的开始位 232 为逻辑真值，且该字节的值为 0x0F。在 x86 的指令中，第一非前置字节的值为 0x0F 表示一具延伸的操作码，因此需要根据其后续字节以决定指令型态。换句话说，无法只从 0x0F 字节以决定指令长度（在一些情形下，可能需要至多到第五字节以决定指令长度）。然而，等到指令高速缓存 102 提供下一列快取数据给 XIBQ104 时，将需要一段时间，例如，可能发生指令高速缓存 102 的失误 (miss)，或指令转译寻找缓冲器 (translation lookaside buffer，TLB) 的失误，因此，需要一种不等待其它指令字节而径行处理的方案。再者，在一些情形下，微处理器 100 必须得到未知长度指令之前的指令，因此如果这些指令未进行处理，则微处理器 100 就要一直等待。因此，需要一种径行处理的方式。

[0116] 图 8 显示图 7 的部分指令格式器 106 的操作流程图。此流程始于步骤 802。

[0117] 于步骤 802，XIBQ 控制逻辑单元 702 检测到 XIBQ104 的底部项目终端的指令跨至指令快取数据串流的另一列，而 XIBQ104 底部项目中的指令不足以让长度解码器 202/ 涟波逻辑单元 204 决定指令长度（及开始 / 结束位 232/234），而决定指令长度所需的后续指令字节尚未置于 XIBQ104NTBE 中，亦即，XIBQ104NTBE 为无效或空白的。接着，流程进入步骤 804。

[0118] 于步骤 804，M 级控制逻辑单元 512 将相应于 XIBQ104 底部项目所产生的涟波逻辑单元 204 的输出 214 载至多工队列 502。然而，M 级控制逻辑单元 512 并不将 XIBQ104 的底部项目移出，因为仍需要决定出未知长度指令的结束位 234。换句话说，对于未知长度的指令，其位于 XIBQ104 底部项目的字节必须保留，当指令的其它字节来到 XIBQ104 时，得以决定出指令长度及结束位。接着，流程进入步骤 806。

[0119] 于步骤 806，前一步骤 804 所载入的输出 214 到达多工队列 502 的底部项目。此时，M 级控制逻辑单元 512 取出所有指令并将其传至 F 级，但不传送未知长度的指令。然而，M 级控制逻辑单元 512 并不移出多工队列 502 的底部项目，因为未知长度的指令的结束位 234 还未得知，且指令的其余字节尚未可得。M 级控制逻辑单元 512 知道未知长度指令的存在，因为该指令不具有效结束位 234。换句话说，已具有效开始位 232 指向指令的第一字节，

但是不具有效结束位 234 指向多工队列 502 的底部项目的字节且 NTBE 为无效。接着，流程进入 808。

[0120] 于步骤 808，M 级控制逻辑单元 512 停止 (stall) 多工队列 502，直到 NTBE 填入有效输出 214。接着，流程进入步骤 812。

[0121] 于步骤 812，XIBQ104 终于自指令高速缓存 102 接收到一列的指令字节 132，其被载至 NTBE 中。该列的指令字节 132 包含未知长度指令的其余字节。接着，流程进入步骤 814。

[0122] 于步骤 814，长度解码器 202/ 涟波逻辑单元 204 针对未知长度指令产生指令长度 222 及开始 / 结束位 232/234。在一实施例中，XIBQ 控制逻辑单元 702 依据指令长度 222 以计算未知长度指令的其余字节数量（其位于步骤 812 载至 XIBQ104 的 NTBE 中）。该其余字节数量于接下来的步骤 818 中是用以决定结束位 234 的位置。接着，流程进入步骤 816。

[0123] 于步骤 816，XIBQ 控制逻辑单元 702 将底部项目移出。然而，M 级控制逻辑单元 512 并不将相应底部项目的涟波逻辑单元 204 的输出 214 加载，因为其根据步骤 804 已置于多工队列 502 中。接着，流程进入步骤 818。

[0124] 于步骤 818，长度解码器 202/ 涟波逻辑单元 204 处理新的 XIBQ104 底部项目（亦即，于步骤 812 所接收的快取数据），且 M 级控制逻辑单元 512 将涟波逻辑单元 204 的输出 214（其包含未知长度指令的结束位 234）载至多工队列 502 的 NTBE 中。接着，流程进入步骤 822。

[0125] 于步骤 822，M 级控制逻辑单元 512 自多工队列 502 的底部项目及 NTBE 取出未知长度指令（以及其它可取出的指令），并传送至 F 级。接着，流程进入步骤 824。

[0126] 于步骤 824，M 级控制逻辑单元 512 将多工队列 502 的底部项目移出。流程结束于步骤 824。

[0127] 根据上述，本实施例的指令格式器 106 即使在 XIBQ(x86 指令字节队列)104 底部项目的相关消息尚未可用的情况下，对于具有可用消息的指令，通过让消息（指令字节、开始 / 结束位及累积前置消息）尽快从 L 级释出，因而得以解决了前述问题。

[0128] [通过前置累积以增进指令的取得]

[0129] 图 9 显示图 5 的多工队列 502 的详细方块图。在图 9 的实施例中，多工队列 502 包含四个项目，分别为底部项目 (bottom entry，BE)、NTBE、底部倒数第三项目 (second-from-bottom entry，SFBE) 及底部倒数第四项目 (third-from-bottom entry，TFBE)。多工队列 502 的每一项目含有十六个字节，每一字节存放一个指令字节及其开始位 232、结束位 234 及累积前置消息 238。如图所示，BE 分别标示为 0 至 15。NTBE 分别标示为 16 至 31。这些标号也显示于图 10。SFBE 分别标示为 32 至 47。

[0130] 图 10 显示图 1 的指令格式器 106 的部分 M 级的方块图。图 10 显示多工队列 502 的累积前置阵列 (accumulated prefix array)1002 及指令字节阵列 (instruction byte array)1004。累积前置阵列 1002 及指令字节阵列 1004 的消息实际上是储存于多工队列 502 的 BE 和 NTBE。然而，多工队列 502 消息的提供是通过导线至选择电路（在一实施例中，其为动态逻辑单元），其包含图 5 的多工器 504/506/508。图 10 仅显示出 I1 多工器 504，然而 I2 多工器 506 及 I3 多工器 508 所接收的输入也如同 I1 多工器 504。指令多工器 504/506/508 为 16:1 多工器。如图 10 所示，I1 多工器 504 的输入分别标示为 0 至 15。每一个 I1 多工器 504 的输入接收十一个指令字节及累积前置消息 238，其中累积前置消息 238 相应于所接

收十一个指令字节的最低位 (lowest order) 字节。该最低位字节为指令字节阵列 1004 的字节号码，其对应至 I1 多工器 504 的输入号码。例如，I1 多工器 504 的输入 8 接收多工队列 502 的字节 8 至 18（亦即 BE 的字节 8-15 及 NTBE 的字节 16-18）及相应字节 8 的累积前置消息 238。I1 多工器 504 接收十一个指令字节的理由为：虽然 x86 指令允许最多十五字节，然非前置字节最多为十一字节，前述实施例仅取得并传送非前置字节至管线的其余部分（亦即，去除前置字节并以累积前置消息 238 置换前置字节），因而可以大量减少管线后续级的解码工作量并让微处理器 100 实现各种好处。

[0131] 图 11 显示图 5 的 M 级控制逻辑单元 512 的方块图。M 级控制逻辑单元 512 包含 2:1 多工器 1114，用以产生指令长度 LEN1 1122，其为通过指令格式器 106 的指令串流的一指令（图 5 的第一指令 I1 524）的指令长度。指令长度 LEN11122 连同第一指令 I 1524 继续通过管线传送并被处理。多工器 1114 根据前一时钟周期中是否有部分长度的情形存在，以选择减法器 1102 的输出或加法器 1116 的输出。多工器 1114 受控于暂存器 1118，其储存一位用以表示前一时钟周期是否存在有部分长度的情形，此将于图 12 至图 14 详述。如果有部分长度情形发生，多工器 1114 选择加法器 1116 的输出；否则，多工器 1114 选择减法器 1102 的输出。加法器 1116 的第一输入为指令剩余长度，标示为剩余 LEN1 1106，其将于图 12 至图 14 详述。M 级控制逻辑单元 512 还包含其它逻辑单元（未显示于图式中），其依据第一指令 I1 524 的结束位 234（其是由多工队列 502 提供给 M 级控制逻辑单元 512）以计算剩余 LEN1 1106。加法器 1116 的第二输入为目前指令的部分长度，标示为部分 LEN 1104，其是由前一时钟周期加载的暂存器所提供，将于图 12 详述。减法器 1102 以前一指令的结束位 234 在多工队列 502 中的字节位置 (END0 1112) 减去第一指令 I1524 的结束位 234 在多工队列 502 中的字节位置 (END1 1108)。值得注意的是，虽然 M 级控制逻辑单元 512 执行如图 11 所示的数学运算，然而 M 级控制逻辑单元 512 不能使用传统加法器 / 减法器，而是以组合逻辑单元来实施。例如，在一实施例中，位是以解码形式来执行的；例如，减法运算可使用布尔 (Boolean)AND-OR 运算。第二指令 I2 526 和第三指令 I3 528 的长度计算所使用的减法器（未显示于图式中）类似于第一指令 I1 524 的减法器，但是分别为 END1 减去 END2 以及 END2 减去 END3。最后，多工队列 502 项目的目前偏移 (offset) 的决定是选择来自多工器 504/506/508 的最后指令最后字节的后一字节。

[0132] 图 12 显示图 1 的指令格式器 106 的部分 M 级的操作流程图。此流程始于步骤 1201。

[0133] 于步骤 1201，新时钟周期开始，且 M 级控制逻辑单元 512 检视多工队列 502 的 BE 及 NTBE（图 9）。接着，流程进入步骤 1202。

[0134] 于步骤 1202，M 级控制逻辑单元 512 控制多工器 504/506/508，将多工队列 502 的 BE 及 NTBE（可能的话）的指令传送至指令格式器 106 的 F 级。如前所述，在一实施例中，M 级可于一时钟周期内取得三指令。由于 x86 指令的长度可为零至十五字节，因此多工队列 502 的底部项目可能存有一至十六个 x86 指令。因此，需要多个时钟周期以取得多工队列 502 的 BE 的所有指令。再者，依据 BE 的最后字节究竟为前置字节、结束字节或其它类型字节，指令可能跨越 BE 和 NTBE，因此，M 级控制逻辑单元 512 在取得指令及移出多工队列 502 的 BE 时，其操作方式会有不同，此将于以下详述。再者，M 级控制逻辑单元 512 计算每一取得 / 传送指令的长度，特别是使用图 11 的逻辑以计算第一指令 I1 524（图 11 的指令长度 LEN1 1122）。如果为前一时钟周期的部分长度（此将于步骤 1212 详述），则 M 级控制逻辑

单元512使用储存的部分LEN1104以计算指令长度LEN1 1122；否则，M级控制逻辑单元512使用减法器1102（图11）以计算指令长度LEN1 1122。接着，流程进入步骤1204。

[0135] 于步骤1204，M级控制逻辑单元512判定是否结束于BE的所有指令都已传送至F级。在一实施例中，于一时钟周期内，M级最多可取得及传送三个指令给F级。因此，如果M级自底部项目取得三指令，且尚有至少另一指令的开始位232在底部项目中，则另一指令必须于下一时钟周期取得。如果结束于BE的所有指令都已传送至F级，则流程进入步骤1206；否则，流程进入步骤1205。

[0136] 于步骤1205，M级控制逻辑单元512不移出BE，使得于下一时钟周期时，M级控制逻辑单元512可以自BE取得及传送更多的指令。流程返回至步骤1201，以进行下一时钟周期的程序。

[0137] 于步骤1206，M级控制逻辑单元512判定BE的最后字节究竟为前置或者为非前置字节。如果BE的最后字节为非前置字节，则流程进入步骤1216；如果BE的最后字节为前置字节，则流程进入步骤1212。

[0138] 于步骤1212，M级控制逻辑单元512计算位于BE最后包含前置字节的指令的部分长度，亦即，从前一指令的结束字节一直到BE的最后字节15之间的前置字节数目，该计算是由M级控制逻辑单元512的数学逻辑单元（未显示于图式中）执行。例如，于图13的例子中，指令b的部分长度为14。位于结束字节和开始字节之间的前置字节是处于“三不管地带”(no-man’s land)，而前置字节于多工队列502中事实上为多余的，因为其内容已经存在于累积前置消息238，其和指令的操作码字节储存于多工队列502内。藉此，如果BE的最后为前置字节且在BE中的所有其它指令于该时钟周期皆已取得，则M级控制逻辑单元512即可将BE(1214)移出（步骤1214），因为，这些前置字节是存在的（其将于接下来的十六字节列当中累积于操作码字节）且M级控制逻辑单元512将前置字节数目储存起来（储存至图11的部分长度暂存器1104）并自多工队列502移出。另一方面，如果BE的最后为非前置字节且其尚未被取得或传送，则M级控制逻辑单元512不能将其自多工队列502移出（参阅步骤1222）。接着，流程进入步骤1214。

[0139] 于步骤1214，M级控制逻辑单元512控制多工队列502以移出BE。流程返回至步骤1201，以进行下一时钟周期的程序。

[0140] 于步骤1216，M级控制逻辑单元512判定BE的最后字节是否为指令的结束字节，亦即，结束位234是否为逻辑真值。如果为是，则流程进入步骤1214；否则，流程进入步骤1218。

[0141] 于步骤1218，M级控制逻辑单元512判定NTBE是否为有效。当取得的最后指令的结束字节位于BE的最后字节（亦即字节15），或者最后字节跨至NTBE且其为有效，则M级控制逻辑单元512移出BE；否则，M级控制逻辑单元512维持BE直到下一时钟周期。如果NTBE为有效，流程进入步骤1214；否则，流程进入步骤1222。

[0142] 于步骤1222，M级控制逻辑单元512不移出BE。此乃因为指令的实际字节（亦即，非前置字节）跨越BE和NTBE，且NTBE为无效。在此情形，M级控制逻辑单元512无法决定指令长度，因为指令的结束位234无法自无效的NTBE得知。流程返回至步骤1201，进行下一时钟周期的程序，以等待NTBE填满有效数据。

[0143] 图13显示图5的多工队列502于连续两个时钟周期的内容，以例示M级的操作。

第一个多工队列 502 内容处于第一时钟周期 0，而第二个多工队列 502 内容处于第二时钟周期 1。图式仅显示出底部的三个项目。于图 13 中，“S”表示开始字节（亦即，开始位 232 为逻辑真值），“E”表示结束字节（亦即，结束位 234 为逻辑真值），“P”表示前置字节（亦即，累积前置消息 238 所表示）。4 个指令分别以 a、b、c、d 来表示，并显示其开始、结束及前置字节。所示字节数目对应至图 9，例如字节 0 至 47，其位于多工队列 502 的 BE、NTBE 及 SFBE。

[0144] 在周期 0 的一开始，BE 的字节 1 包含有指令 a 的结束字节 Ea，且 BE 的字节 2 至 15 包含有十四个指令 b 的前置字节 Pb。因为指令 b 开始于 BE，但其开始字节是位于 NTBE 而不是 BE，其部分长度计算为十四字节。NTBE 及 SFBE 的内容为无效的，亦即 X86 指令字节队列 104 和长度解码器 202/ 涟波逻辑单元 204 尚未提供指令串流的快取数据及其相关消息（例如开始位 232、结束位 234 以及累积前置消息 238）至除了 BE 之外的其它项目。

[0145] 于周期 0 时，M 级控制逻辑单元 512 检视 BE 和 NTBE 的内容（图 12 的步骤 1201）并传送指令 a 至 F 级（步骤 1202）。再者，M 级控制逻辑单元 512 计算指令 a 的长度，其等于指令 a 的结束字节位置和前一指令的结束字节位置之间的差值。最后，由于结束于 BE 的所有指令（指令 a）已传送（步骤 1204）且 BE 的最后一字节（字节 15）为前置字节（步骤 1206），M 级控制逻辑单元 512 计算指令 b 的部分长度为十四字节，将其储存于部分 LEN 1104 暂存器（步骤 1212）。最后，M 级控制逻辑单元 512 从多工队列 502 将 BE 移出（步骤 1214）。

[0146] 由于步骤 1214 于周期 0 进行了移出且移入另外十六字节列的涟波逻辑单元 204 输出 214，因而开始周期 1，此时 BE 包含：位于字节 0 的指令 b 的开始字节（Sb）及结束字节（Eb）（亦即，指令 b 的非前置字节仅有单一字节）；位于字节 1 至 5 的指令 c 的五个前置字节（Pc）；位于字节 6 的指令 c 的开始字节（Sc）；位于字节 8 的指令 c 的结束字节（Ec）；位于字节 9 的指令 d 的开始字节（Sd）；及位于字节 15 的指令 d 的结束字节（Ed）。

[0147] 于周期 1 时，M 级控制逻辑单元 512 检视 BE 和 NTBE 的内容（步骤 1201）且传送指令 b、c 及 d 至 F 级（步骤 1202）。再者，M 级控制逻辑单元 512 计算以下各项：指令 b 的长度（LEN1 1122）（步骤 1202）（在此例子中为十五字节），其等于部分 LEN 1104（在此例子中为十四字节）加上指令 b 的剩余长度（在此例子中为一字节）；指令 c 的长度（在此例子中为八字节），其等于指令 c 的结束字节位置和指令 b 的结束字节位置两者的差值；及指令 d 的长度（在此例子中为七字节），其等于指令 d 的结束字节位置和指令 c 的结束字节位置两者的差值。再者，由于所有结束于 BE 的指令（指令 b，c，d）都已传送（步骤 1204）且 BE 的最后字节（字节 15）为非前置字节（步骤 1206）且 BE 的最后字节为结束字节（步骤 1216），因此 M 级控制逻辑单元 512 从多工队列 502 将 BE 移出（步骤 1214）。

[0148] 根据图 13 所示的实施例，通过累积指令 b 的累积前置消息 238 至其操作码且储存指令 b 的部分 LEN 1104，使得指令格式器 106 可将含有指令 b 的前置字节的 BE 移出，并于下一时钟周期从多工队列 502 取得及传送最多三个指令。如果没有累积前置消息 238 及储存部分 LEN 1104，这将是不可能的（亦即，指令 c 和 d 无法与指令 b 在相同周期内取得及传送，而是必须于下一时钟周期进行）。通过使得微处理器的功能单元具足够指令可处理，可减少微处理器 100 资源的使用。

[0149] 图 14 显示图 5 的多工队列 502 于连续两个时钟周期的内容，以例示 M 级的操作。

图 14 的例子类似于图 13 的例子；然而，指令的位置及多工队列 502 的进入及离开时序则有差异。

[0150] 于周期 0 的一开始，BE 位于字节 1 包含有指令 a 的结束字节 (Ea)，且位于字节 2 至 15 包含有指令 b 的十四个前置字节 (Pb)。另外，由于指令 b 开始于 BE，但是其开始字节却是位于 NTBE，因此部分 LEN 1104 计算为 14。NTBE 包含：位于字节 16 的指令 b 的开始字节 (Sb) 及指令 b 的结束字节 (Eb)（亦即，指令 b 除了前置字节外，仅为单一字节）；位于字节 17-21 的指令 c 的五个前置字节 (Pc)；位于字节 22 的指令 c 的开始字节 (Sc)；位于字节 27 的指令 c 的结束字节 (Ec)；位于字节 28-30 的指令 d 的三个前置字节 (Pd)；及位于字节 31 的指令 d 的开始字节 (Sd)。SFBE 包含：位于字节 41 的指令 d 的结束字节 (Ed)，及位于字节 42 的指令 e 的开始字节 (Se)。

[0151] 于周期 0 时，M 级控制逻辑单元 512 检视 BE 和 NTBE 的内容（图 12 的步骤 1201）并传送指令 a 至 F 级（步骤 1202）。再者，M 级控制逻辑单元 512 计算指令 a 的长度，其等于指令 a 的结束字节位置和前一指令的结束字节位置之间的差值。最后，由于结束于 BE 的所有指令（指令 a）已传送（步骤 1204）且 BE 的最后一字节（字节 15）为前置字节（步骤 1206），M 级控制逻辑单元 512 计算指令 b 的部分长度为十四字节，将其储存于部分 LEN1104 暂存器（步骤 1212）。最后，M 级控制逻辑单元 512 从多工队列 502 将 BE 移出（步骤 1214）。

[0152] 由于步骤 1214 于周期 0 进行了移出，因而开始周期 1，此时 BE 包含周期 0 时的 NTBE 的内容，而 NTBE 包含周期 0 时的 SFBE 的内容。

[0153] 于周期 1 时，M 级控制逻辑单元 512 检视 BE 和 NTBE 的内容（步骤 1201）且传送指令 b、c 及 d 至 F 级（步骤 1202）。再者，M 级控制逻辑单元 512 计算以下各项：指令 b 的长度 (LEN1 1122)（步骤 1202）（在此例子中为十五字节），其等于部分 LEN 1104（在此例子中为十四字节）加上指令 b 的剩余长度（在此例子中为一字节）；指令 c 的长度（在此例子中为十一字节），其等于指令 c 的结束字节位置和指令 b 的结束字节位置两者的差值；及指令 d 的长度（在此例子中为十四字节），其等于指令 d 的结束字节位置和指令 c 的结束字节位置两者的差值。再者，由于所有结束于 BE 的指令（指令 b、c、d）都已传送（步骤 1204）且 BE 的最后字节（字节 15）为非前置字节（步骤 1206）且 BE 的最后字节非为结束字节（步骤 1216）且 NTBE 为有效（步骤 1218），因此 M 级控制逻辑单元 512 从多工队列 502 将 BE 移出（步骤 1214）。

[0154] 根据图 14 所示的实施例，指令格式器 106 可于一时钟周期内，将含有最多四十个指令字节的三个指令取得并传送出去，如图 15 所示。

[0155] [不良分支预测的检测、标示及累积，用以快速处理指令串流]

[0156] 再参阅图 1，当提取单元 126 输出目前提取地址 142 用以自指令高速缓存 102 提取一指令字节列并提供给 XIBQ104 时，BTAC128 也同时得到该目前提取地址 142。如果目前提取地址 142 命中 (hit)BTAC128，则表示先前在此提取地址有一分支指令曾被执行过；因此，BTAC128 可预测是否有分支指令发生 (taken)，如果为是，则 BTAC128 也预测了预测目标地址 146。特别的是，BTAC128 是在微处理器 100 从指令字节串流取得或解码分支指令之前即进行预测。因此，BTAC128 所预测的分支指令可能并未存在于取出的指令字节的高速缓存列中，亦即，BTAC128 作了不良的预测，造成微处理器 100 分支错误。值得注意的是，此不良预测并不等同于不正确预测。由于程序执行具动态性质，例如分支指令的状态码或状态数

据的值的改变，因此所有分支预测器于本质上都有可能会预测错误。然而，此处的不良预测表示 BTAC128 所预测的高速缓存列不同，或者高速缓存列相同但高速缓存列内的内容已经改变。之所以发生这些情形的理由，如美国专利 7,134,005 所描述，理由有下列几种：由于 BTAC128 仅储存部分的地址标签 (tag) 而非完整地址标签，因而造成标签混淆 (aliasing)；由于 BTAC128 仅储存虚拟 (virtual) 地址标签而非物理地址，因而造成虚拟混淆；及自发性修改码 (self-modifying code)。当此情形发生时，微处理器 100 必须确定未将不良预测指令及后续因为不良预测指令而错误取得的错误指令传送出去。

[0157] 如果对于一指令字节其分支发生指示 154（图 1）为逻辑真值但是事实上并非为指令的第一字节，如图 16 所示，即表示 BTAC128 作了不良预测因而造成微处理器 100 的分支错误。如前所述，BTAC128 所提供的真值分支发生指示 154 表示 BTAC128 认为该指令字节为分支指令的第一字节（亦即操作码），且提取单元 126 根据 BTAC128 所预测的预测目标地址 146 进行分支。

[0158] 不良 BTAC 预测的决定方法系等待，直到个别的指令从指令字节串流中取得且长度为已知，并扫描每一指令的非第一字节以检视其分支发生指示 154 是否为真。然而，此种检查方法太慢，因为其需要很多的屏蔽 (masking) 及移出，且需将每一字节的结果经由逻辑或 (OR) 运算，因此会造成时序问题。

[0159] 为了避免时序问题，本发明实施例累积分支发生指示 154 所提供的消息，其为涟波逻辑单元 204 执行程序的一部分，且于 M 级取得指令后，使用这些累积消息。特别的是，涟波逻辑单元 204 检测状态并将指示符传递下去直到指令的最后字节，其检视单一字节，亦即指令的最后字节。当从 M 级取得指令时，决定一指令是否为不良指令，亦即，该指令是否要被涵盖于指令串流并继续沿着管线传送下去。

[0160] 图 17 显示涟波逻辑单元 204 输出 214 的组成信号。图 17 所示的涟波逻辑单元 204 输出信号类似于图 2 所示，但对于每一指令字节额外增加了不良 BTAC 位 1702，其将于以下详述。此外，涟波逻辑单元 204 输出包含：一信号，当其为逻辑真值时，表示相应的指令字节为 BTAC128 所预测的分支指令的第一字节，然而 BTAC128 所预测的分支指令将不会发生；及另一信号，其表示前一字节为指令的结束字节。

[0161] 图 18 显示图 1 的微处理器 100 的操作流程图。此流程始于步骤 1802。

[0162] 于步骤 1802，BTAC（分支目标地址高速缓存）128 预测于提取单元 126 所提供的目前提取地址 142 所指的高速缓存列中，存在一分支指令，且该分支指令将会发生。BTAC（分支目标地址高速缓存）128 还预测分支指令的预测目标地址 146。因此，XIBQ104 于目前提取地址 142 所指处的指令高速缓存 102 接收十六指令字节当中的第一列，且接着于预测目标地址 146 所指处的指令高速缓存 102 接收十六指令字节当中的第二列。接着，流程进入步骤 1804。

[0163] 于步骤 1804，XIBQ104 储存每一分支发生指示 154（图 1）连同于步骤 1802 所接收的二列相应的指令字节。接着，流程进入步骤 1806。

[0164] 于步骤 1806，长度解码器 202 和涟波逻辑单元 204 处理指令字节的第一列，并检测指令字节含有逻辑真值的分支发生指示 154 但该字节并非指令的第一字节的情形，如图 16 所示的错误情形。换句话说，涟波逻辑单元 204 知道指令字节的十六组列中哪一字节为第一字节，用以设定结束位 234。据此，相应每个指令的第一非前置字节的涟波逻辑单元 204

检视分支发生指示 154 的逻辑真值及检测该情形。接着，流程进入步骤 1808。

[0165] 于步骤 1808，当检测到指令的非第一字节的真值分支发生指示 154 为逻辑真值，涟波逻辑单元 204 设定该指令字节的不良 BTAC 位 1702 为逻辑真值。此外，涟波逻辑单元 204 将真值的不良 BTAC 位 1702 从其字节位置传递至十六字节列的其余字节。再者，如果指令的结束字节未出现于指令字节的第一列，则涟波逻辑单元 204 更新状态（例如正反器）（未显示于图式中），用以表示目前列中对一指令发生不良的 BTAC（分支目标地址高速缓存）128 预测。接着，当涟波逻辑单元 204 处理指令字节的第二列时，由于状态为真，涟波逻辑单元 204 对于指令字节第二列的所有字节设定其不良 BTAC 位 1702。接着，流程进入步骤 1812。

[0166] 于步骤 1812，对于指令字节的第一及第二列，多工队列 502 储存涟波逻辑单元 204 的输出 214，包含不良 BTAC 位 1702，并连同各个的指令字节一起储存。接着，流程进入步骤 1814。

[0167] 于步骤 1814，M 级控制逻辑单元 512 发现相应于指令字节的不良 BTAC 位 1702 为逻辑真值且该指令字节的结束位 234 也为逻辑真值（亦即，检测到不良 BTAC（分支目标地址高速缓存）128 预测的情形）。因此，M 级控制逻辑单元 512 通过清除相应的有效位 534/536/538 而放弃传送发生不良情形的指令及其后续指令至 F 级。然而，如果在发生不良情形的指令之前有一指令，则该指令为有效且被传送至 F 级。如前所述，真值的不良 BTAC 位 1702 传递至发生不良情形的指令的结束字节，将使得 M 级控制逻辑单元 512 得以只检视单一字节，亦即，结束位 234 所指的字节，因而明显减少时序的限制。接着，流程进入 1816。

[0168] 于步骤 1816，微处理器 100 让 BTAC（分支目标地址高速缓存）128 的错误项目变为无效。此外，微处理器 100 清除 XIBQ104 及多工队列 502 的所有内容并让提取单元 126 更新目前提取地址 142，用以自 BTAC（分支目标地址高速缓存）128 产生不良预测处重新取得指令字节。于重新取得时，BTAC（分支目标地址高速缓存）128 不会产生不良预测，因为不良项目已被清除，亦即，于重新取得时，BTAC（分支目标地址高速缓存）128 将预测分支不会发生。在一实施例中，步骤 1816 执行于指令格式器 106 的 F 级，及 / 或指令转译器 112。流程结束于步骤 1816。

[0169] [x86 指令长度的有效决定]

[0170] 决定 x86 指令长度是非常复杂的，其描述于英特尔 IA-32 架构软件开发手册 (Intel IA-32Architecture Software Developer's Manual)，第 2A 集的第二章 :指令集参考 (Instruction Set Reference)，A-M。指令总长度系为下列之和 :前置字节的数目（如果有的话）、操作字节的数目 (1、2 或 3)、ModR/M 字节出现与否、SIB 字节出现与否、地址位移 (displacement) 长度（如果有的话）及立即 (immediate) 数据的长度（如果有的话）。以下为 x86 指令的特性或要求，其足以影响长度（前置除外）的决定 :

[0171] 操作码字节的数目为 :

[0172] 3，如果前二字节为 0F 38/3A

[0173] 2，如果第一字字节为 0F，且第二字字节不为 38/3A

[0174] 1，其它情形

[0175] ModR/M 字节是否出现决定于操作码，如下 :

[0176] 如果为三字节操作码，则 ModR/M 为强制的

[0177] 如果为一字节或二字节操作码，则检视操作码字节

[0178] SIB 字节是否出现决定于 ModR/M 字节。

[0179] 位移是否出现决定于 ModR/M 字节。

[0180] 位移尺寸决定于 ModR/M 字节及目前地址尺寸 (AS)。

[0181] 立即数据是否出现决定于操作码字节。

[0182] 立即数据的尺寸决定于操作码字节、目前操作码尺寸 (OS)、目前 AS 及 REX.W 前置；特别的是，ModR/M 字节不会影响立即数据尺寸。

[0183] 如果没有 ModR/M 字节，则没有 SIB、位移或立即数据。

[0184] 当决定指令长度时，指令操作码及 ModR/M 字节仅有五种形式：

[0185] 操作码

[0186] 0F+ 操作码

[0187] 操作码 +ModR/M

[0188] 0F+ 操作码 +ModR/M

[0189] 0F+38/3A+ 操作码 +ModR/M

[0190] 图 19 显示图 2 的长度解码器 202 的详细方块图。图 2 显示了十六个长度解码器 202。图 19 显示一代表性长度解码器 202，标示为 n。如图 2 所示，每一长度解码器 202 对应至指令字节串流 134 的一个字节。换句话说，长度解码器 0 对应至指令字节 0，长度解码器 1 对应至指令字节 1，一直到长度解码器 15 对应至指令字节 15。长度解码器 202 包含可编程逻辑阵列 (Programmable Logic Array，PLA)1902、4:1 多工器 1906 及加法器 1904。

[0191] PLA 1902 接收图 2 所示的地址尺寸 (AS)、操作数尺寸 (OS) 及 REX.W 值 218。AS 代表地址尺寸、OS 代表操作数尺寸，且 REX.W 值表示 REX.W 前置的出现。PLA 1902 还接收相应的指令字节 134(其标示以 n) 及高一阶的指令字节 134(其标示以 n+1)。例如，PLA 31902 接收指令字节 3 及 4。

[0192] PLA 1902 产生 immLen 值 1916，其提供给加法器 1904 的第一输入。immLen 值 1916 介于 1 和 9(含) 之间，其值为下列之和 :操作码字节数目及立即数据的尺寸 (0、1、2、4、8)。PLA 1902 于决定 immLen 值 1916 时，是假设该二指令字节 134 为指令的前二操作码字节，并依据二操作码字节 (如果不是 0F 则为一操作码字节)、地址尺寸 (AS)、操作数尺寸 (OS) 及 REX.W 值 218 以产生 immLen 值 1916。

[0193] PLA 1902 产生 eaLen 值 1912，其提供给三个低阶长度解码器 202 的多工器 1906。eaLen 值 1912 介于 1 和 6(含) 之间，其值为下列之和 :ModR/M 字节数目 (PLA 假设 ModR/M 字节的存在)、SIB 字节数目 (0 或 1) 及位移尺寸 (0、1、2、4)。PLA 1902 于决定 eaLen 值 1912 时，是假设第一指令字节 134 为 ModR/M 字节，并依据 ModR/M 字节和地址尺寸 (AS)218 以产生 eaLen 值 1912。

[0194] 多工器 1906 的其中一个输入接收零值。多工器 1906 的其它三个输入接收来自三个高阶 PLA 1902 的 eaLen 值 1912。多工器 1906 选择其中一个输入用以提供 eaLen 值 1918 作为输出，其再提供给加法器 1904 的第二输入。在一实施例中，为了减少传递延迟，可不使用前述的多工器 1906，各个 eaLen 值 1912 被输入至加法器 1904，其中 eaLen 值 1912 为三态线或 (tri-statewired-OR) 信号。

[0195] 加法器 1904 将 immLen 值 1916 及被选到的 eaLen 值 1918 加总以产生图 2 所示的

最终指令长度 222。

[0196] PLA 1902 产生控制信号 1914 以控制多工器 1906，其根据前述五种形式进行检测如下：

[0197] 1. 对于以下所示的不具 ModR/M 字节的指令形式，则选择零值：

[0198] 仅操作码，或

[0199] 0F+ 操作码

[0200] 2. 对于以下指令形式，则选择 PLA n+1：

[0201] 操作码 +ModR/M

[0202] 3. 对于以下指令形式，则选择 PLA n+2：

[0203] 0F+ 操作码 +ModR/M

[0204] 4. 对于以下指令形式，则选择 PLA n+3：

[0205] 0F+38/3A+ 操作码 +ModR/M

[0206] 图 20 显示十六个长度解码器 202 的配置。PLA 15（可编程逻辑阵列）1902 接收指令字节 15 及前一列的指令字节 0，而多工器 151906 接收三个 PLA1902（图未示）的 eaLen 值 1912，其中该三个 PLA 1902 分别检视前一列的指令字节 0/1、1/2 及 2/3。

[0207] 前述每一 PLA 1902 每一次检视二字节的好处在于可大量减少所需的全及项 (minterm) 数目，因而得以减小晶圆上的逻辑电路的尺寸。此设计提供总全及项数目的减少和时序要求所允许的延迟两者之间的平衡选择。

[0208] 图 21 显示图 20 的长度解码器 202 的操作流程图。此流程始于步骤 2102。

[0209] 于步骤 2102，对于来自 XIBQ104 的每一指令字节 134，相应的 PLA 1902 检视二指令字节 134，亦即相应的指令字节 134 及下一指令字节 134。例如，PLA 3（可编程逻辑阵列）1902 检视指令字节 3 和 4。接着，流程同时进入步骤 2104 和 2106。

[0210] 于步骤 2104，每一 PLA 1902 假设二指令字节 134 为指令的前二操作码字节，且依据该二指令字节 134、操作数尺寸 (OS)、地址尺寸 (AS)、及 REX.W 值以产生 immLen 值 1916。详而言之，immLen 值 1916 为下列之和：操作码字节的数目 (1、2 或 3) 和立即数据的尺寸 (0、1、2、4 或 8)。接着，流程进入步骤 2114。

[0211] 于步骤 2106，每一 PLA 1902 假设第一指令字节 134 为 ModR/M 字节，且依据 ModR/M 字节及地址尺寸 (AS) 以产生 eaLen 值 1918，并提供 eaLen 值 1918 给次三个低阶多工器 1906。详而言之，eaLen 值 1918 为下列之和：ModR/M 字节数目 (1)、SIB 字节 (0 或 1) 和位移的尺寸 (0、1、2、4)。接着，流程进入步骤 2108。

[0212] 于步骤 2108，每一多工器 1906 接收零输入及自三高阶 PLA 1902 接收的 eaLen 值 1912。例如，PLA 3（可编程逻辑阵列）1902 自 PLA 4、5、6（可编程逻辑阵列）1902 接收 eaLen 值 1912。接着，流程进入步骤 2112。

[0213] 于步骤 2112，每一 PLA 1902 产生控制信号 1914 至相应的多工器 1906，并依据前述五种形式以选择其中一输入。接着，流程进入步骤 2114。

[0214] 于步骤 2114，每一加法器 1904 将 immLen 值 1916 加至多工器 1906 所选择的 eaLen 值 1918，以产生指令长度 222。接着，流程进入步骤 2116。

[0215] 于步骤 2116，如果出现有 LMP，则 L 级对于含有 LMP 的每一指令花费额外的一个时钟周期，如前述图式所示，特别是图 1 至图 4。

[0216] 以上所述仅为本发明的实施例而已，并非用以限定本发明的权利要求范围。熟悉计算机领域的人士在未脱离发明所揭示的精神下所完成的等效改变或修饰，均应包含在上述的权利要求范围内。例如，可使用软件以启动功能、制造、建立模型、仿真、描述及 / 或测试所揭露的装置及方法。其达成可使用程序语言（例如 C、C++）、硬件描述语言 (HDL)，其包含 Verilog HDL、VHDL 及其它程序。该软件可置于计算机可使用媒体，例如半导体、磁盘或光盘（例如 CD-ROM、DVD-ROM）。所揭露的装置及方法实施例可包含于知识产权核心 (IPcore)，例如微处理器核心（例如置于 HDL）并转换为硬件以制成集成电路。再者，所揭露的装置及方法实施例可使用硬件和软件的组合来实施。因此，本发明范围不限定于任何例示实施例，而应以权利要求范围及其等效范围来定义。详而言之，本发明可实施于微处理器装置内，该微处理器可用于一般计算机内。最后，本领域技术人员可使用所揭露的概念及特定实施例作为基础以设计或修改成其它架构，用以实现相同目的，其仍未脱离本发明的权利要求范围。

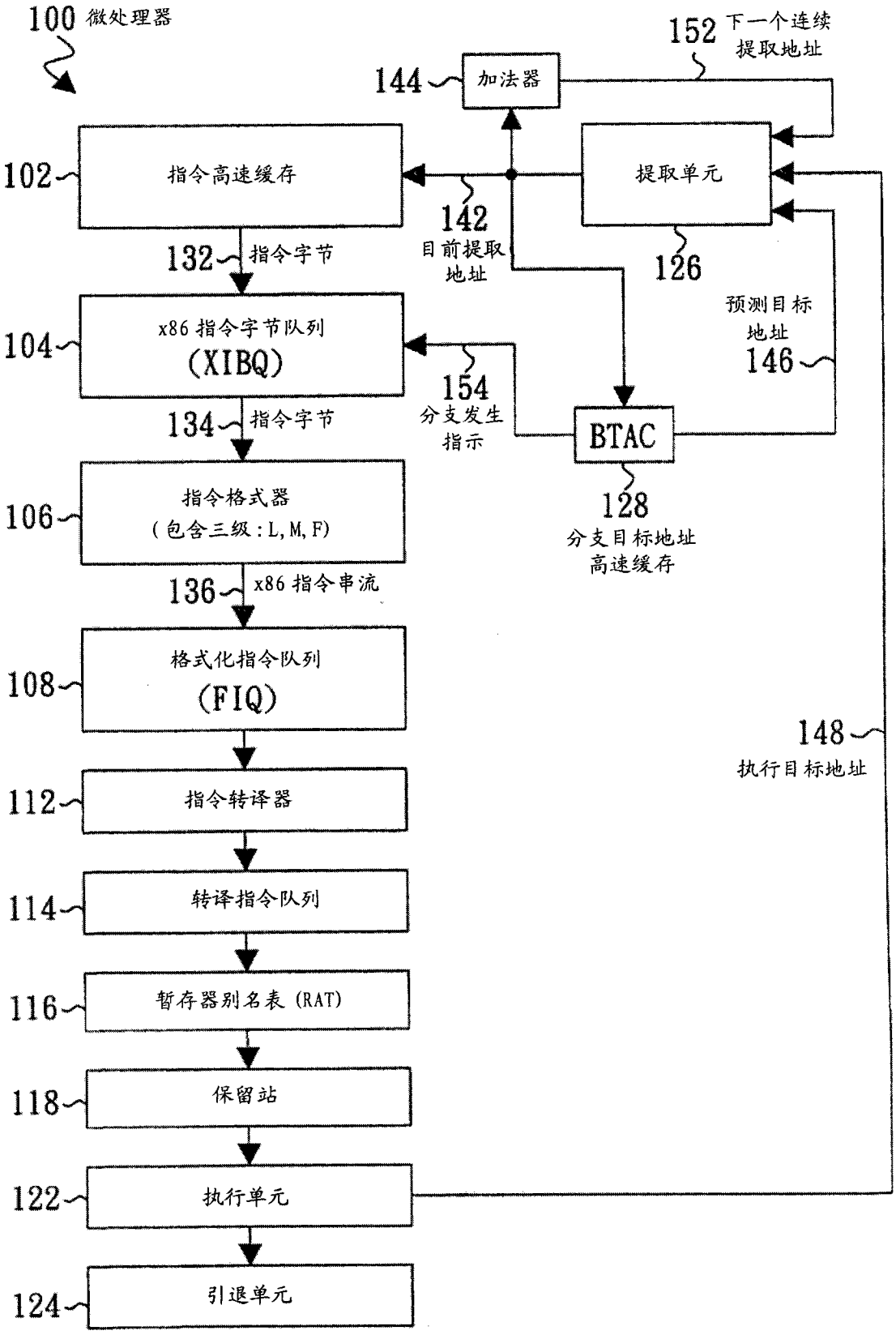
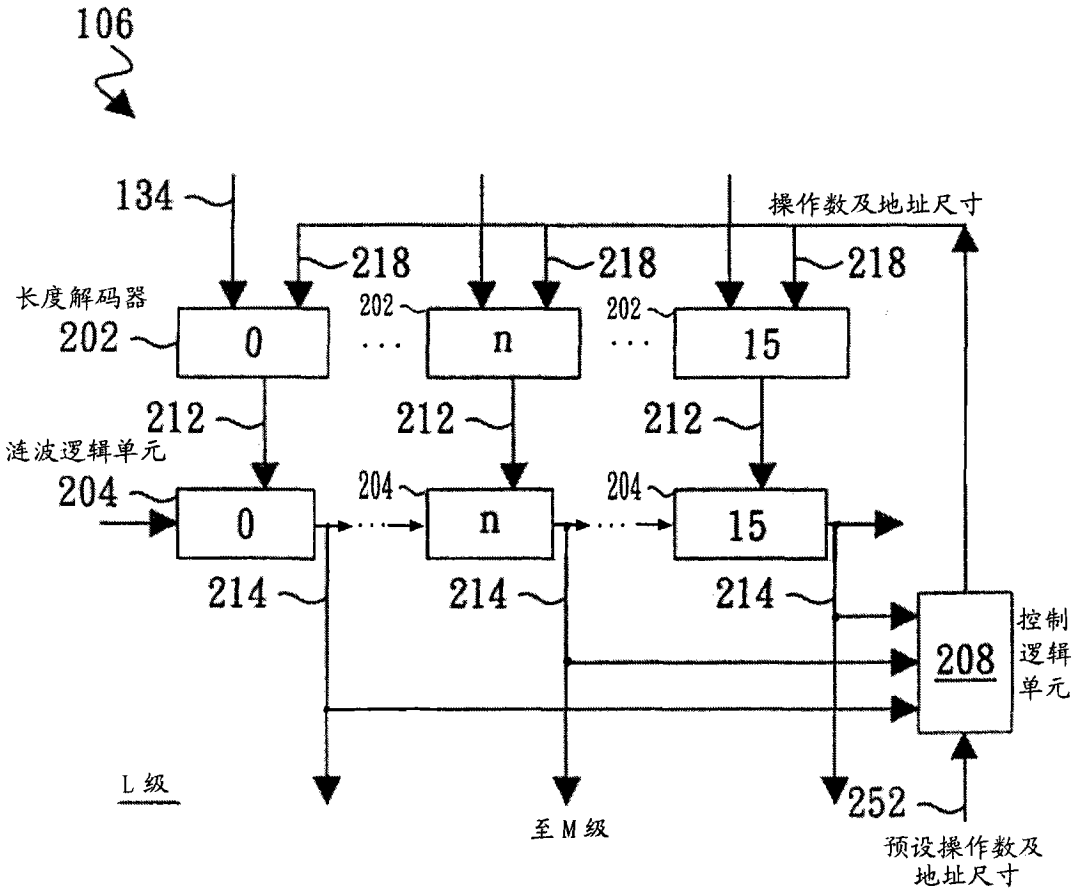

100 微处理器
152 下一个连续提取地址
144 加法器
102 指令高速缓存
提取单元
142 目前提取地址
126
132 指令字节
预测目标地址
146
104 x86 指令字节队列 (XIBQ)
154 分支发生指示
BTAC
134 指令字节
106 指令格式器 (包含三级：L, M, F)
128 分支目标地址高速缓存
136 x86 指令串流
108 格式化指令队列 (FIQ)
148 执行目标地址
112 指令转译器
114 转译指令队列
116 暂存器别名表 (RAT)
118 保留站
122 执行单元
124 引退单元

图 1


106
134
操作数及地址尺寸
218
218
218
长度解码器
202
0
202
n
202
15
212
212
212
涟波逻辑单元
204
0
204
n
204
15
214
214
214
208
控制逻辑单元
L 级
至 M 级
252
预设操作数及地址尺寸

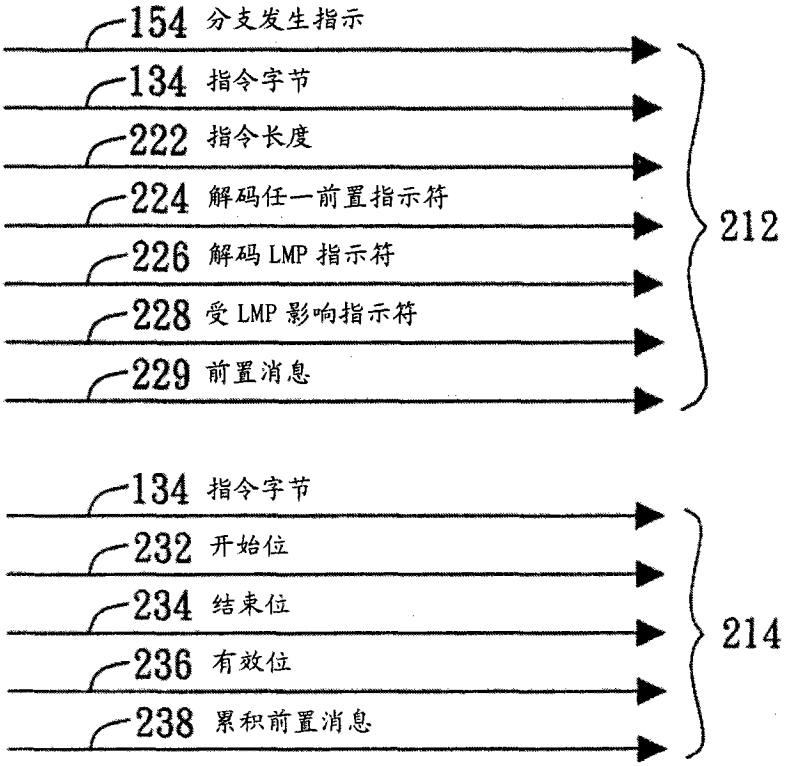

154 分支发生指示
134 指令字节
222 指令长度
224 解码任一前置指示符
226 解码 LMP 指示符
228 受 LMP 影响指示符
229 前置消息
212
134 指令字节
232 开始位
234 结束位
236 有效位
238 累积前置消息
214

图 2

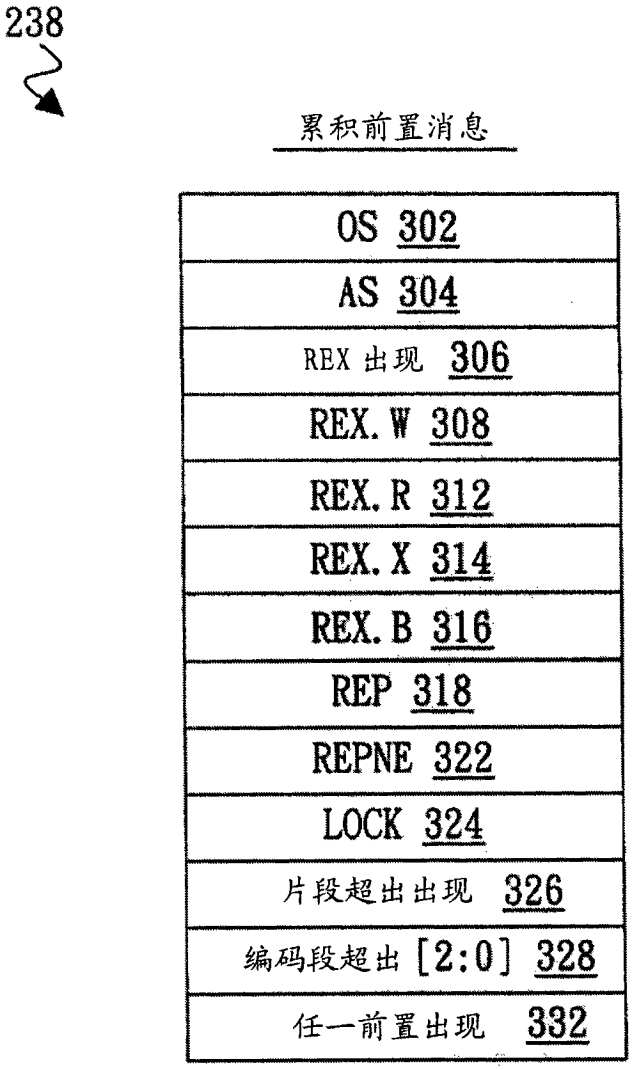

238
累积前置消息
OS 302
AS 304
REX 出现 306
REX.W 308
REX.R 312
REX.X 314
REX.B 316
REP 318
REPNE 322
LOCK 324
片段超出出现 326
编码段超出 [2:0] 328
任一前置出现 332

图 3

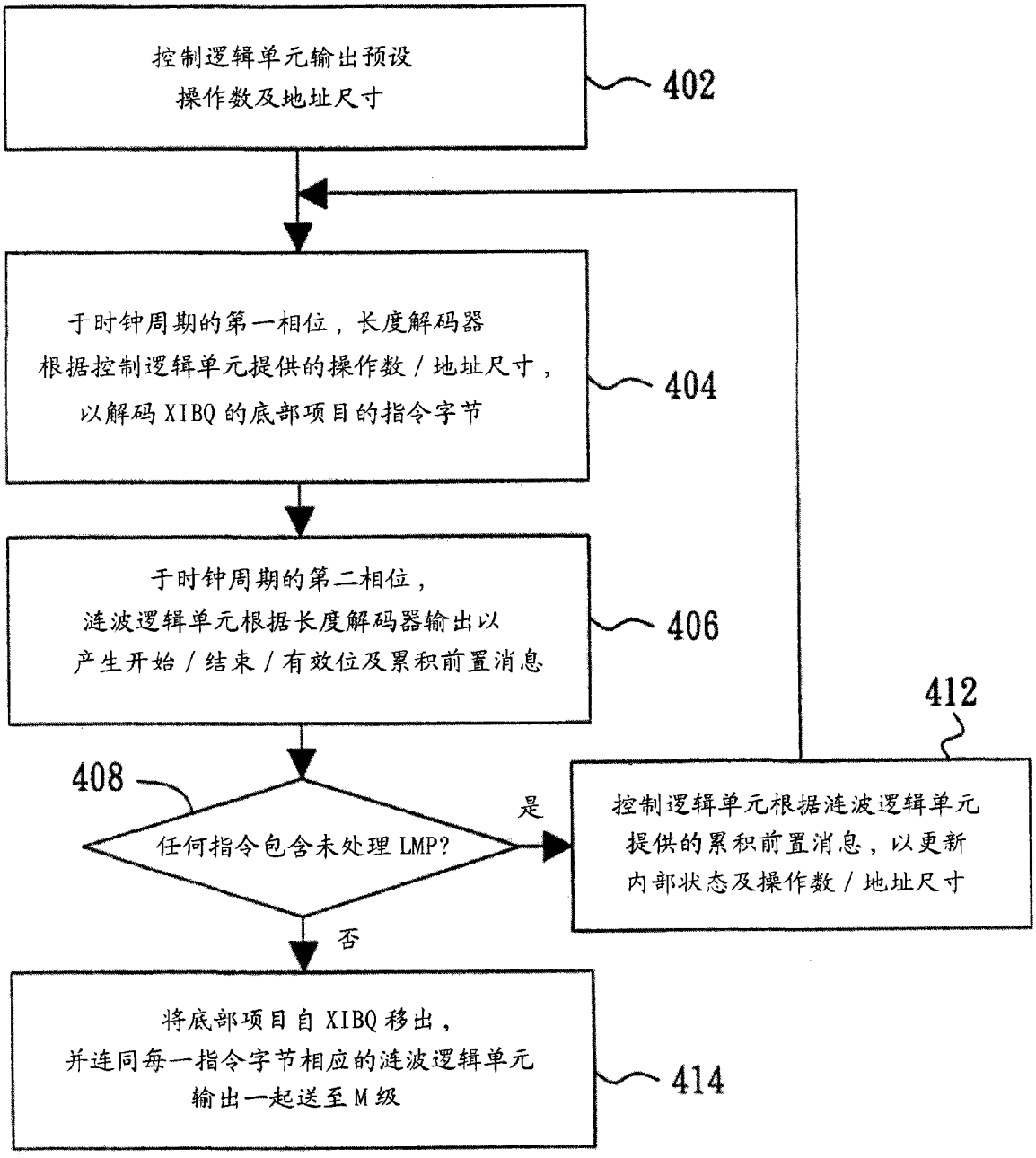

控制逻辑单元输出预设
操作数及地址尺寸
402
于时钟周期的第一相位，长度解码器
根据控制逻辑单元提供的操作数 / 地址尺寸，
以解码 XIBQ 的底部项目的指令字节
404
于时钟周期的第二相位，
涟波逻辑单元根据长度解码器输出以
产生开始 / 结束 / 有效位及累积前置消息
406
412
408
任何指令包含未处理 LMP?
是
控制逻辑单元根据涟波逻辑单元
提供的累积前置消息，以更新
内部状态及操作数 / 地址尺寸
否
将底部项目自 XIBQ 移出，
并连同每一指令字节相应的涟波逻辑单元
输出一起送至 M 级
414

图 4

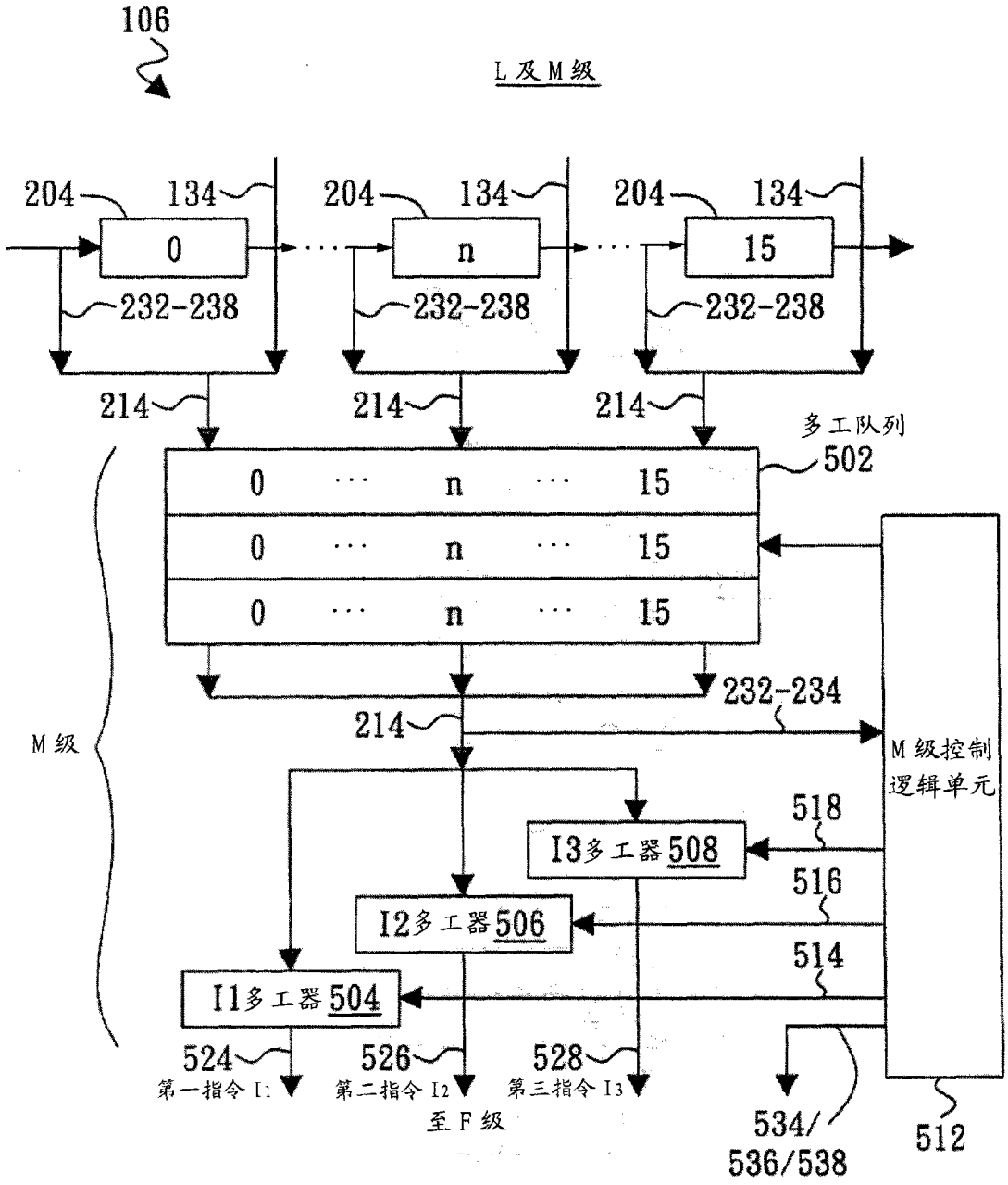

106
L及M级
204
134
0
n
15
232-238
214
多工队列
502
M级
232-234
M级控制
逻辑单元
518
I3多工器508
516
I2多工器506
514
I1多工器504
524
526
528
第一指令 I1
第二指令 I2
第三指令 I3
至F级
534/
536/538
512

图 5

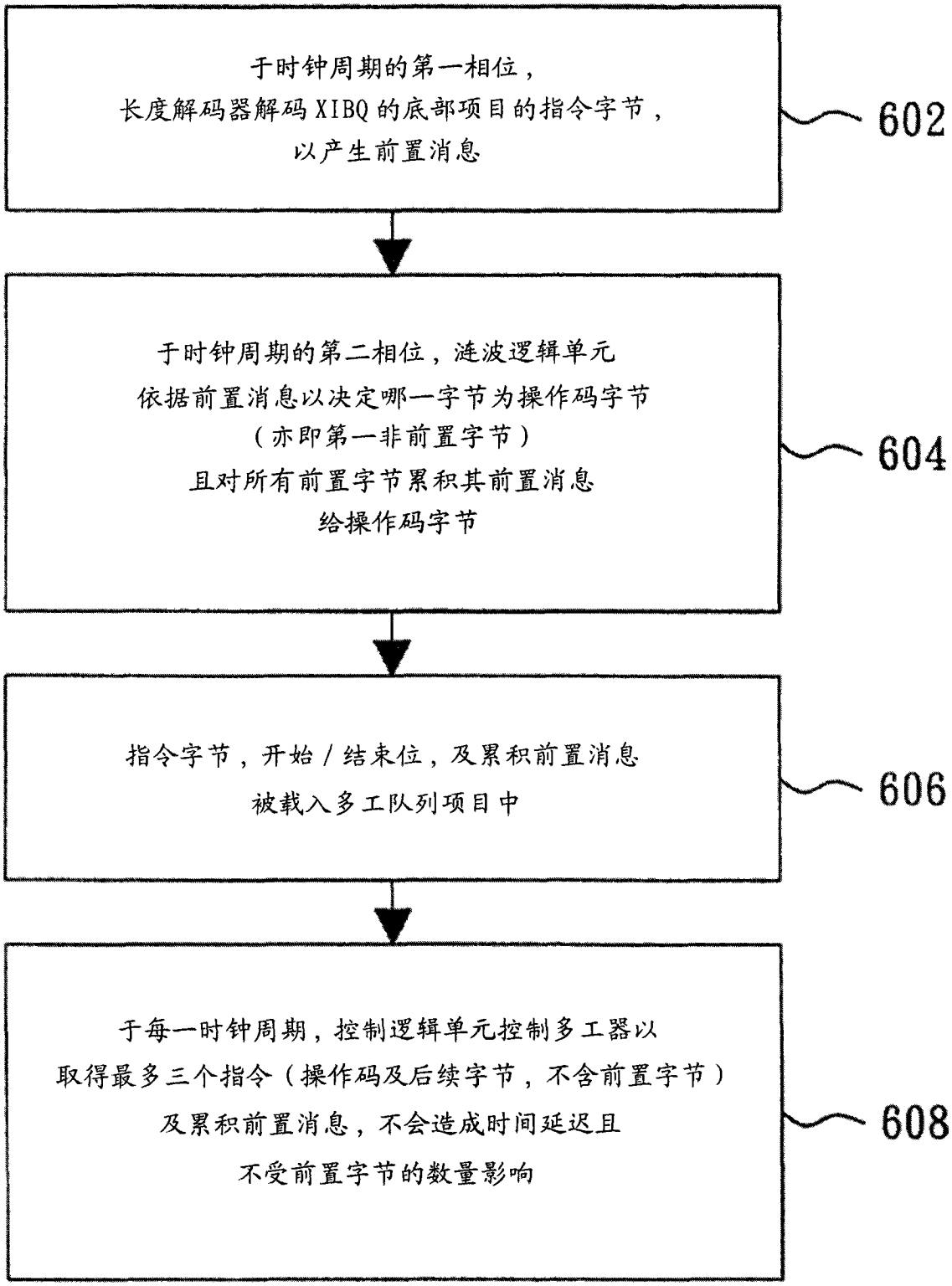

于时钟周期的第一相位，
长度解码器解码 XIBQ 的底部项目的指令字节，
以产生前置消息
602
于时钟周期的第二相位，涟波逻辑单元
依据前置消息以决定哪一字节为操作码字节
（亦即第一非前置字节）
且对所有前置字节累积其前置消息
给操作码字节
604
指令字节，开始 / 结束位，及累积前置消息
被载入多工队列项目中
606
于每一时钟周期，控制逻辑单元控制多工器以
取得最多三个指令（操作码及后续字节，不含前置字节）
及累积前置消息，不会造成时间延迟且
不受前置字节的数量影响
608

图 6

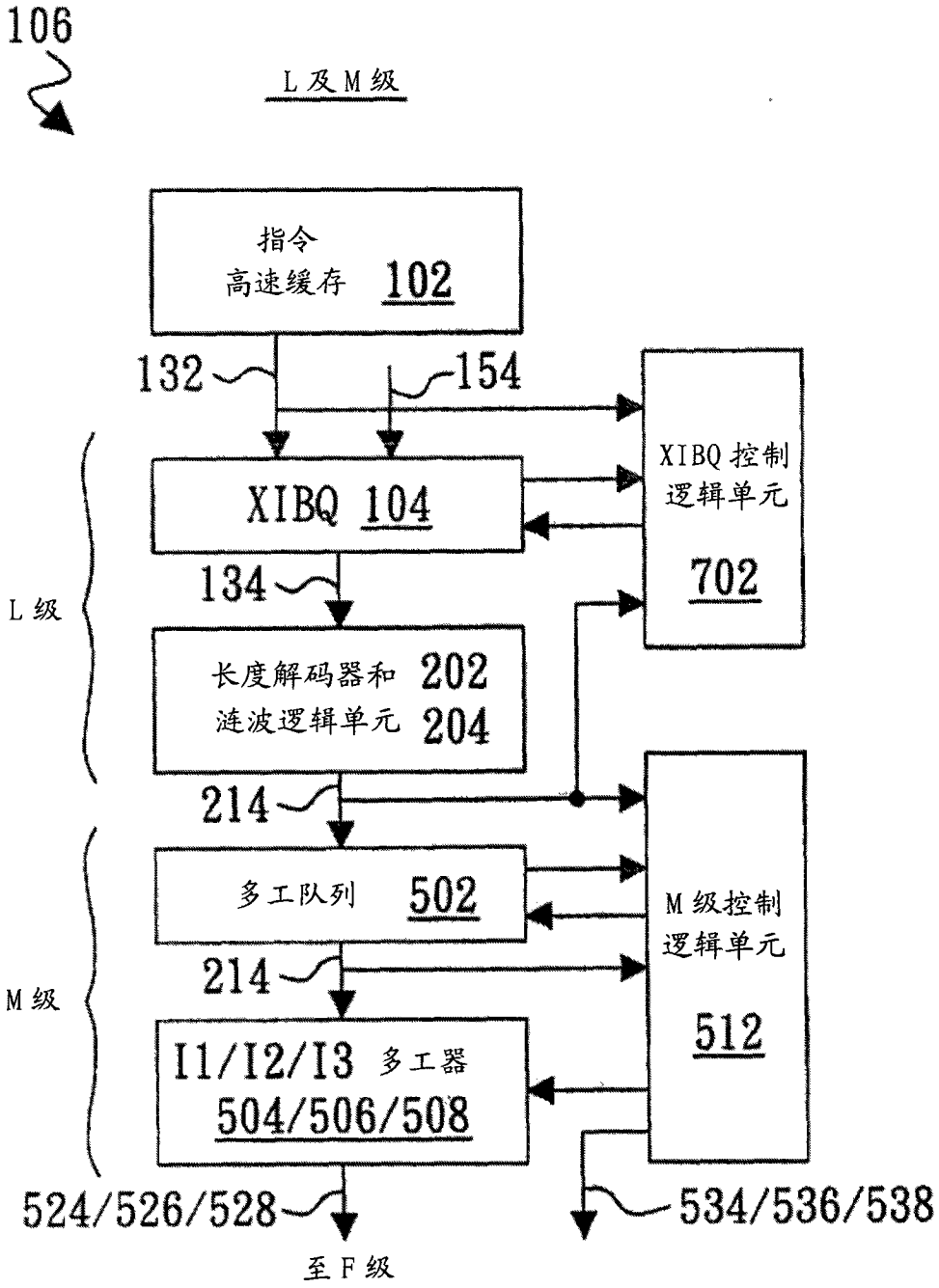

106
L 及 M 级
指令
高速缓存 102
132
154
XIBQ 104
XIBQ 控制
逻辑单元
702
L 级
134
长度解码器和 202
涟波逻辑单元 204
214
多工队列 502
M 级控制
逻辑单元
512
214
M 级
I1/I2/I3 多工器
504/506/508
524/526/528
534/536/538
至 F 级

图 7

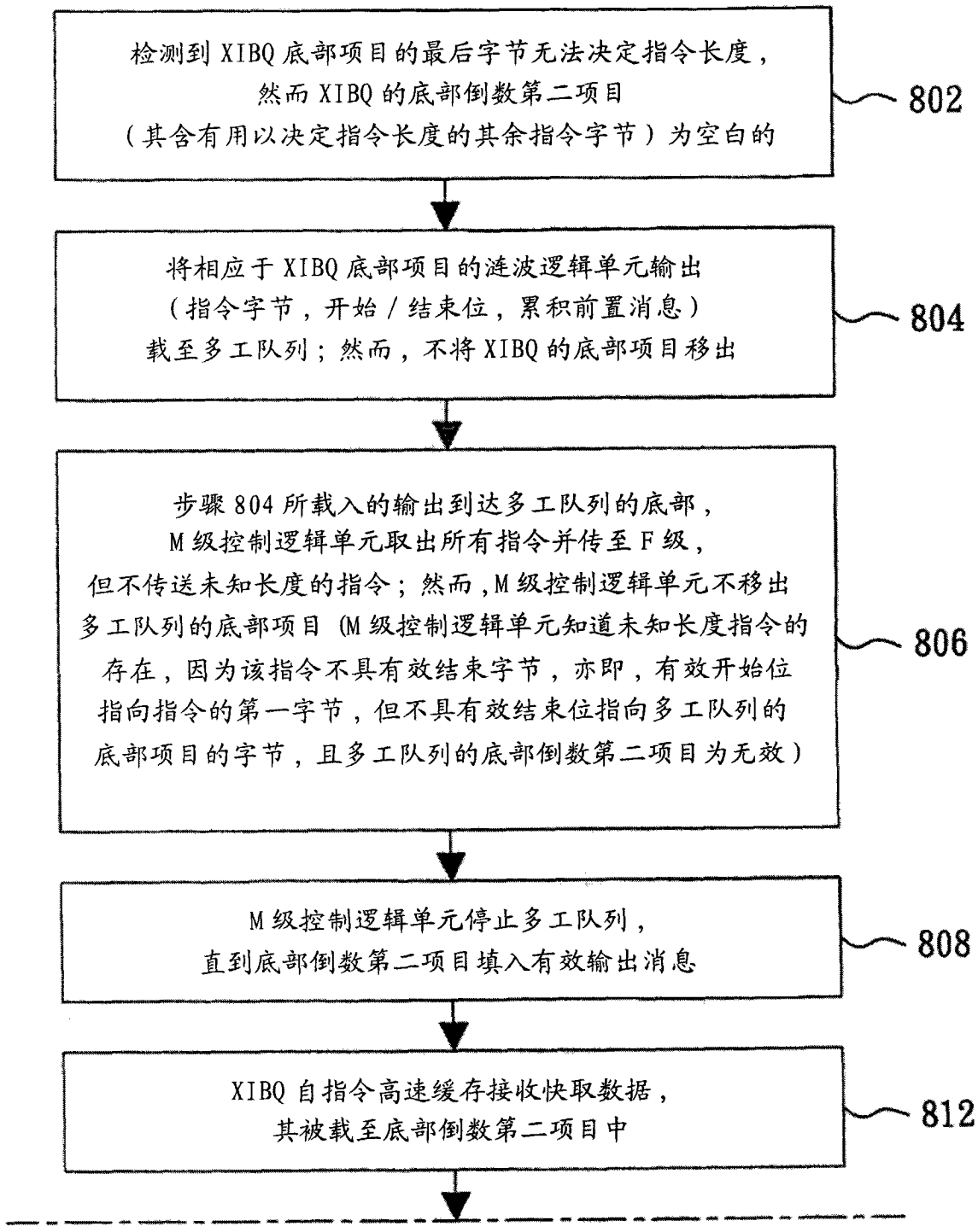

检测到 XIBQ 底部项目的最后字节无法决定指令长度，
然而 XIBQ 的底部倒数第二项目
（其含有用以决定指令长度的其余指令字节）为空白的
802
将相应于 XIBQ 底部项目的涟波逻辑单元输出
（指令字节，开始 / 结束位，累积前置消息）
载至多工队列；然而，不将 XIBQ 的底部项目移出
804
步骤 804 所载入的输出到达多工队列的底部，
M 级控制逻辑单元取出所有指令并传至 F 级，
但不传送未知长度的指令；然而，M 级控制逻辑单元不移出
多工队列的底部项目（M 级控制逻辑单元知道未知长度指令的
存在，因为该指令不具有效结束字节，亦即，有效开始位
指向指令的第一字节，但不具有效结束位指向多工队列的
底部项目的字节，且多工队列的底部倒数第二项目为无效）
806
M 级控制逻辑单元停止多工队列，
直到底部倒数第二项目填入有效输出消息
808
XIBQ 自指令高速缓存接收快取数据，
其被载至底部倒数第二项目中
812

图 8a

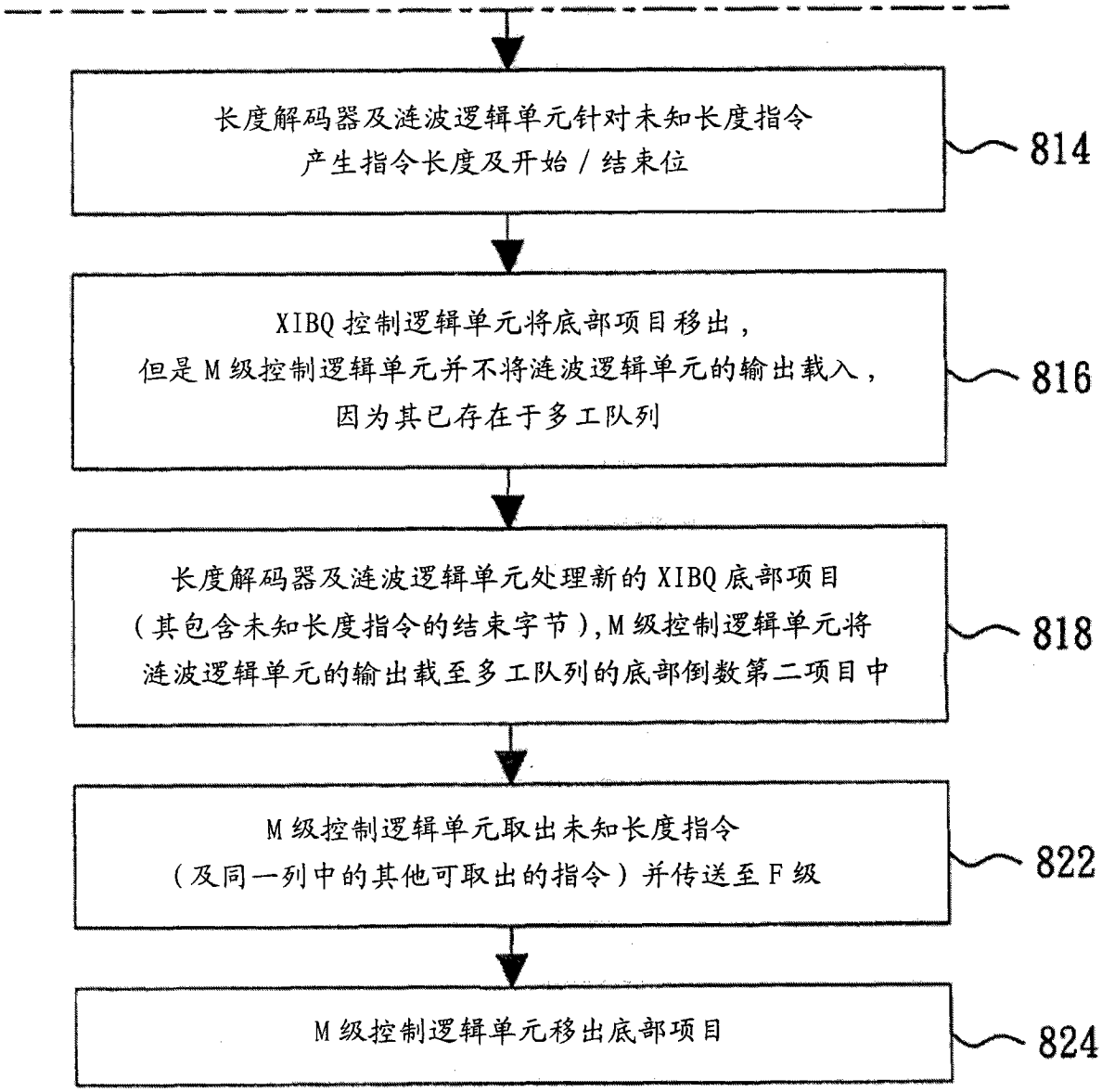

长度解码器及涟波逻辑单元针对未知长度指令
产生指令长度及开始 / 结束位
814
XIBQ 控制逻辑单元将底部项目移出，
但是 M 级控制逻辑单元并不将涟波逻辑单元的输出载入，
因为其已存在于多工队列
816
长度解码器及涟波逻辑单元处理新的 XIBQ 底部项目
（其包含未知长度指令的结束字节），M 级控制逻辑单元将
涟波逻辑单元的输出载至多工队列的底部倒数第二项目中
818
M 级控制逻辑单元取出未知长度指令
（及同一列中的其他可取出的指令）并传送至 F 级
822
M 级控制逻辑单元移出底部项目
824

图 8b

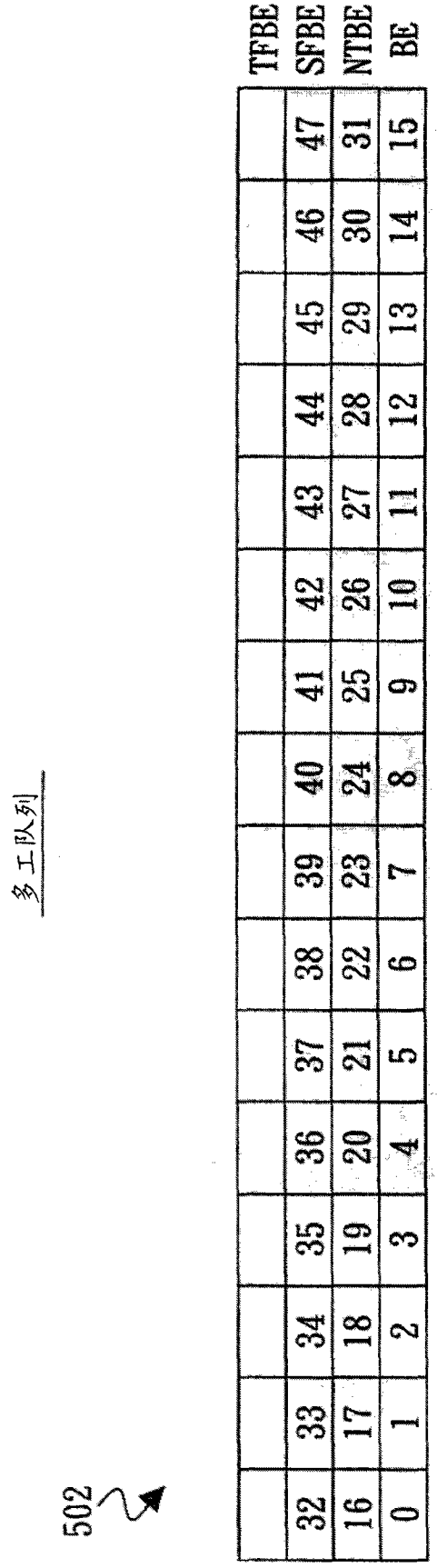

多工队列
502
TFBE
SFBE
NTBE
BE
32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

图 9

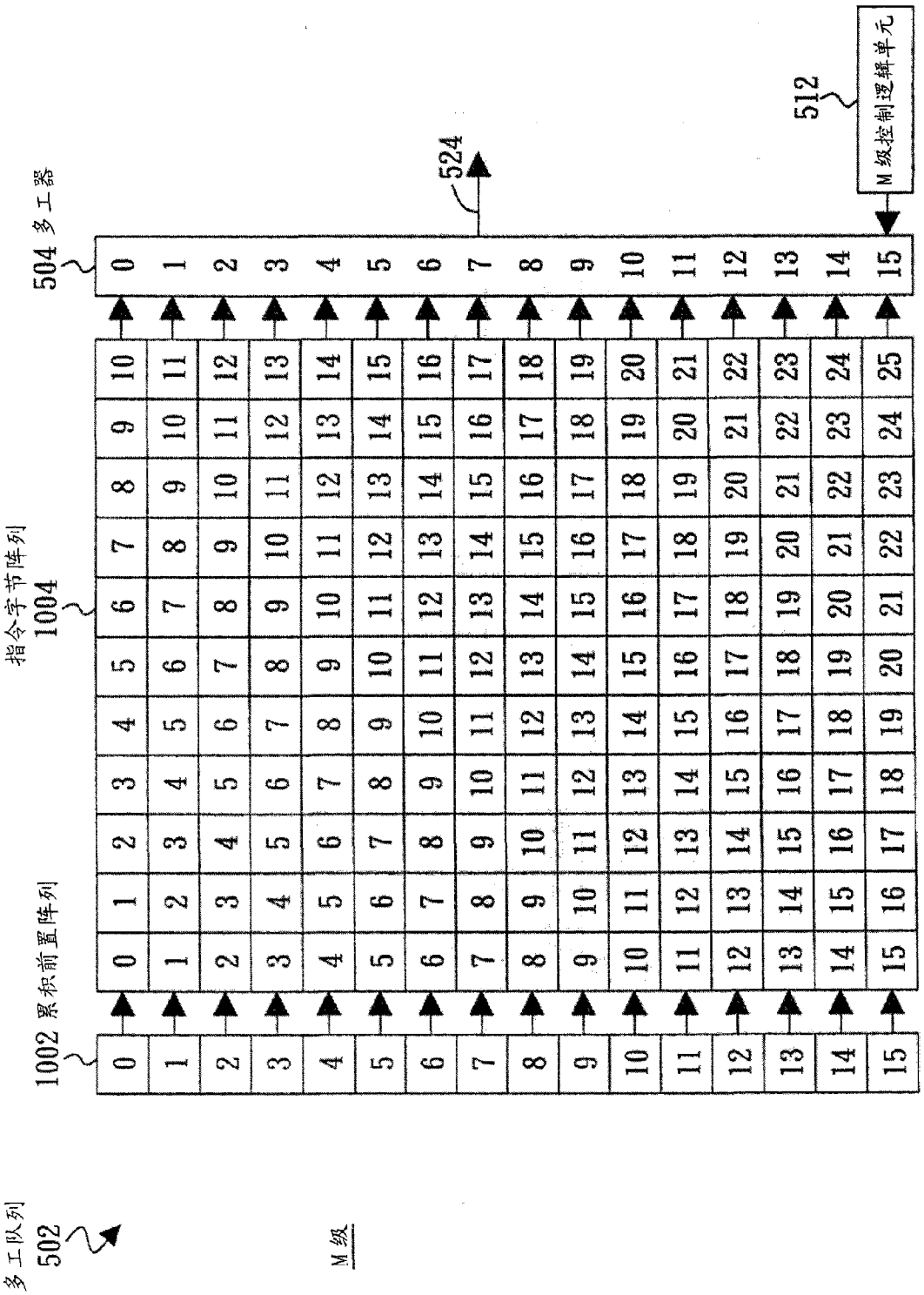


图 10

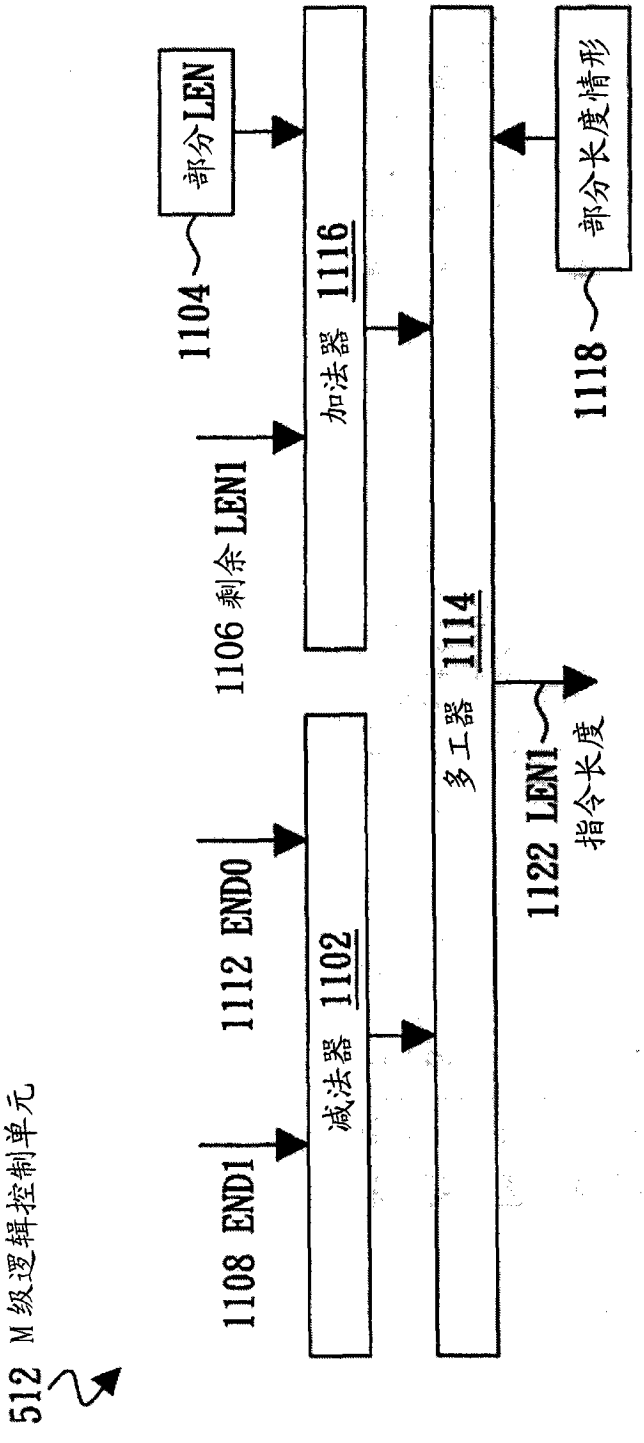

512 M 级逻辑控制单元
1108 END1
1112 END0
减法器 1102
1104 部分 LEN
1106 剩余 LEN1
加法器 1116
多工器 1114
1118 部分长度情形
1122 LEN1 指令长度

图 11

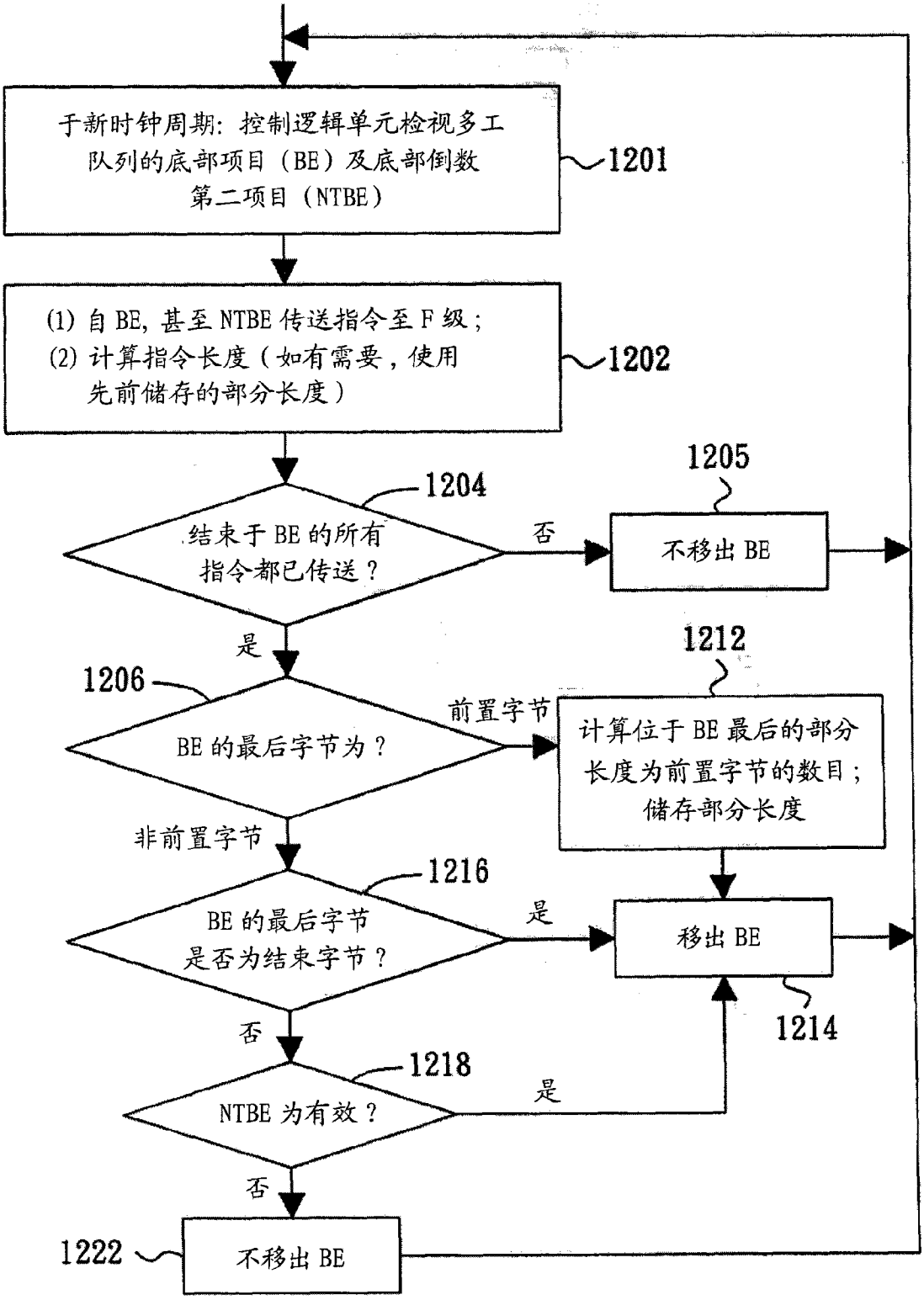

于新时钟周期：控制逻辑单元检视多工队列的底部项目（BE）及底部倒数第二项目（NTBE）
1201
(1) 自 BE，甚至 NTBE 传送指令至 F 级；
(2) 计算指令长度（如有需要，使用先前储存的部分长度）
1202
1204
结束于 BE 的所有指令都已传送？
否
1205
不移出 BE
是
1206
BE 的最后字节为？
前置字节
1212
计算位于 BE 最后的部分长度为前置字节的数目；储存部分长度
非前置字节
1216
BE 的最后字节是否为结束字节？
是
移出 BE
1214
否
1218
NTBE 为有效？
是
否
1222
不移出 BE

图 12

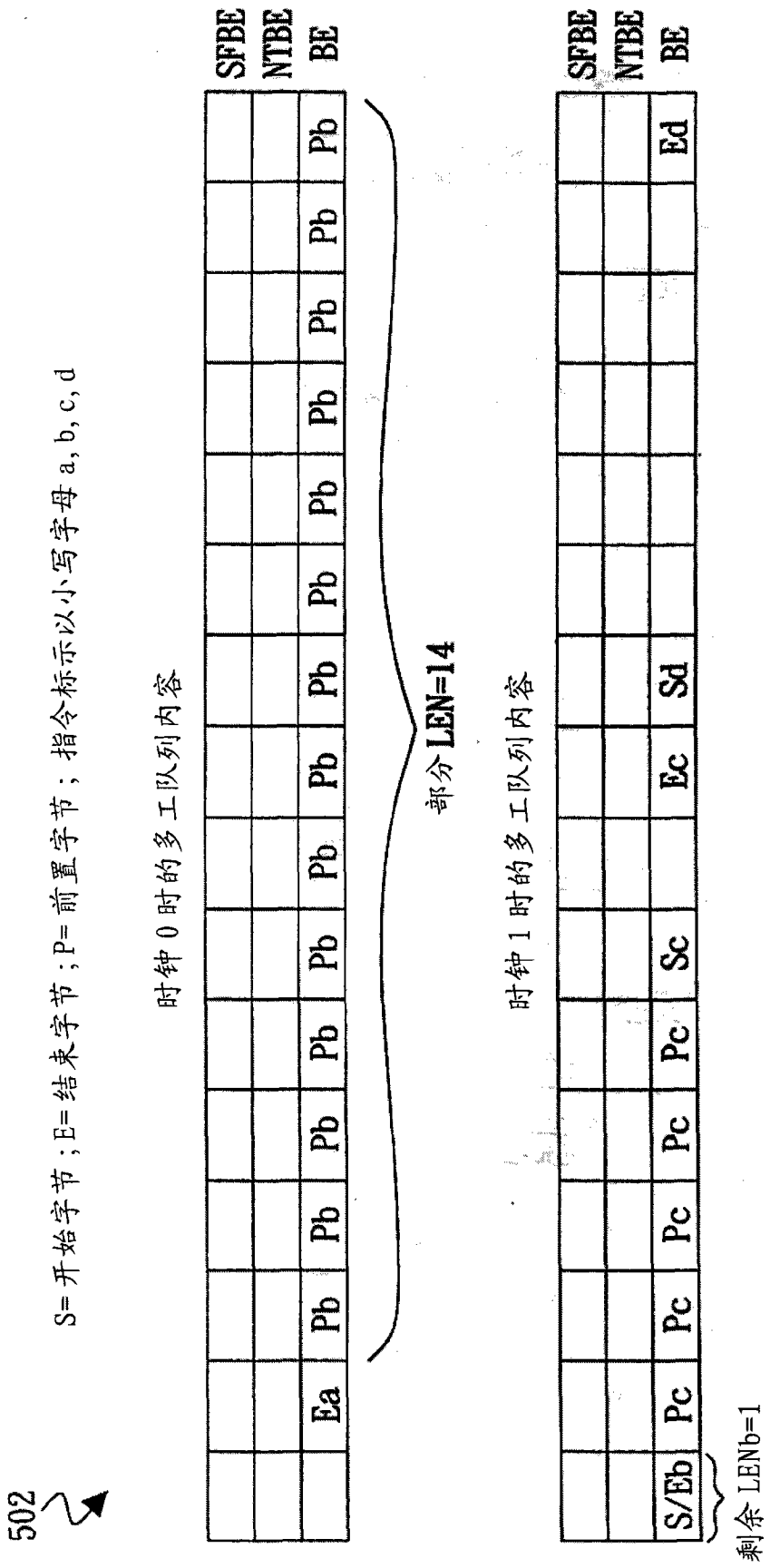

502
S= 开始字节；E= 结束字节；P= 前置字节；指令标示以小写字母 a, b, c, d
时钟 0 时的多工队列内容
SFBE
NTBE
BE
Ea
Pb
Pc
Sc
Ec
Sd
Ed
S/Eb
部分 LEN=14
时钟 1 时的多工队列内容
剩余 LENb=1

图 13

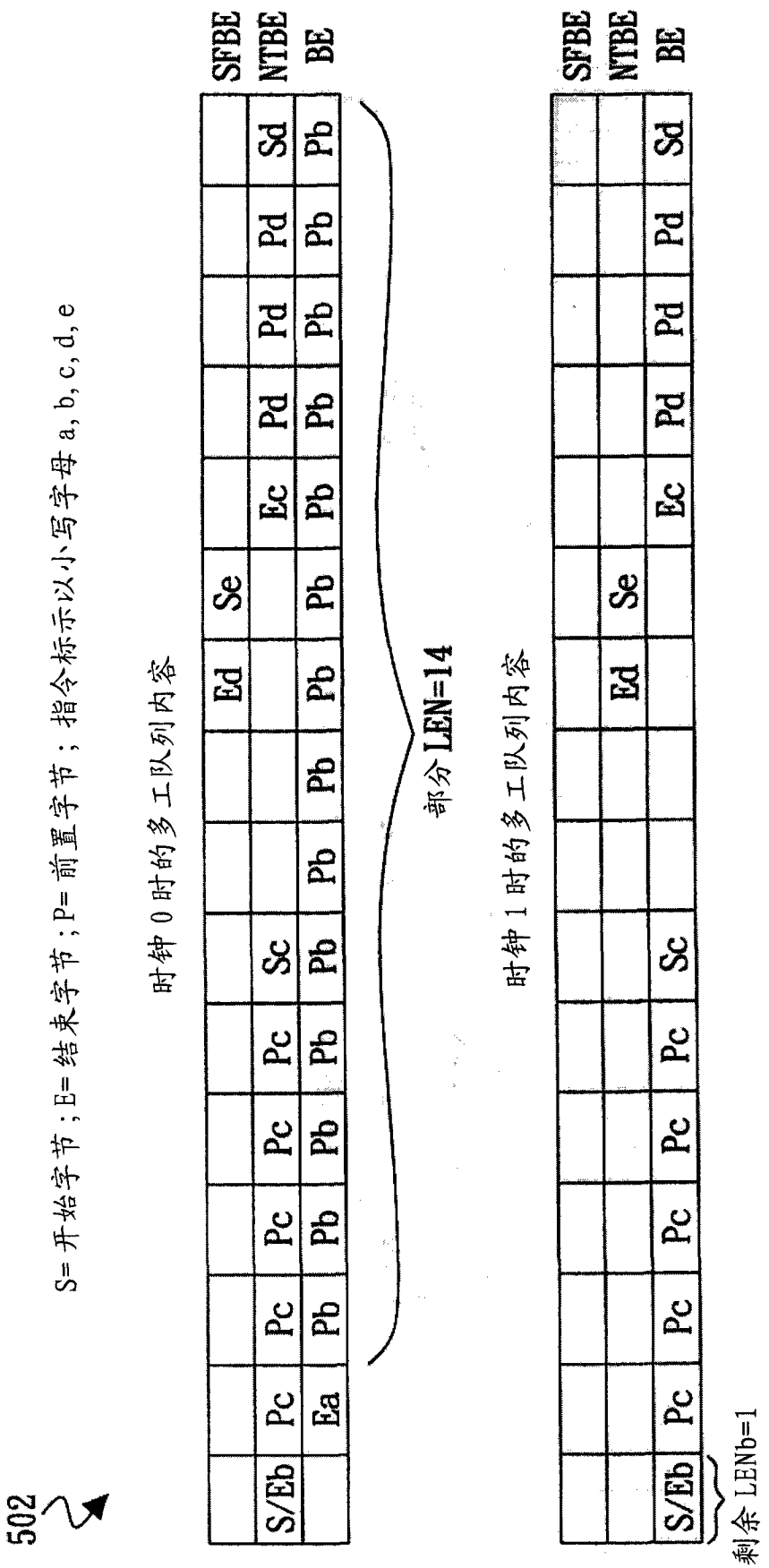

502
S= 开始字节；E= 结束字节；P= 前置字节；指令标示以小写字母 a, b, c, d, e
时钟 0 时的多工队列内容
SFBE
NTBE
BE
S/Eb Pc Pc Pc Pc Sc Ed Se Ec Pd Pd Pd Sd
Ea Pb Pb Pb Pb Pb Pb Pb Pb Pb Pb Pb Pb
部分 LEN=14
时钟 1 时的多工队列内容
SFBE
NTBE
BE
Ed Se Ec Pd Pd Pd Sd
S/Eb Pc Pc Pc Pc Sc Ec Pd Pd Pd Pb
剩余 LENb=1

图 14

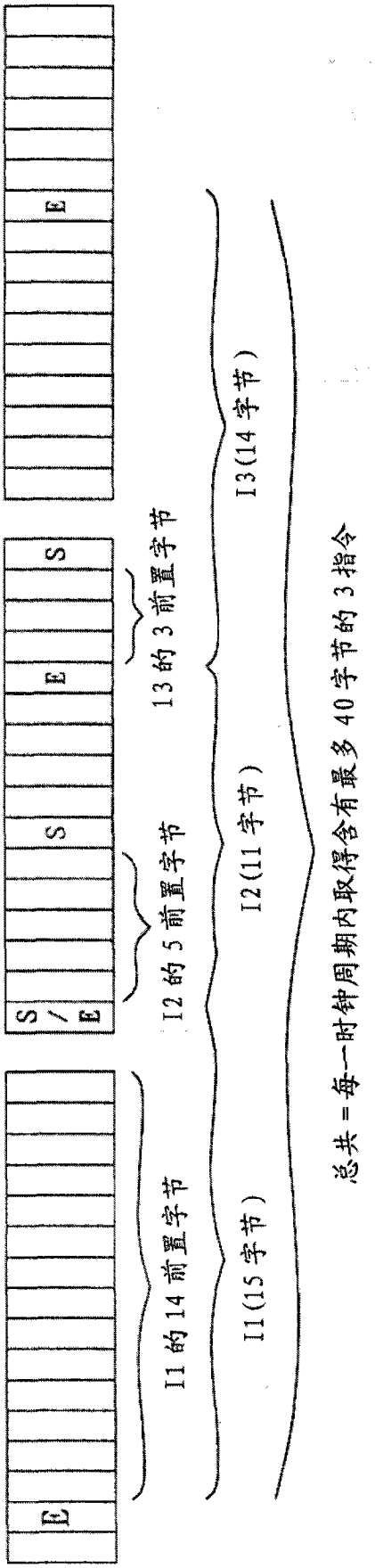

E
S / E
S
E
S
E
I1 的 14 前置字节
I2 的 5 前置字节
13 的 3 前置字节
I1(15 字节)
I2(11 字节)
I3(14 字节)
总共 = 每一时钟周期内取得含有最多 40 字节的 3 指令

图 15

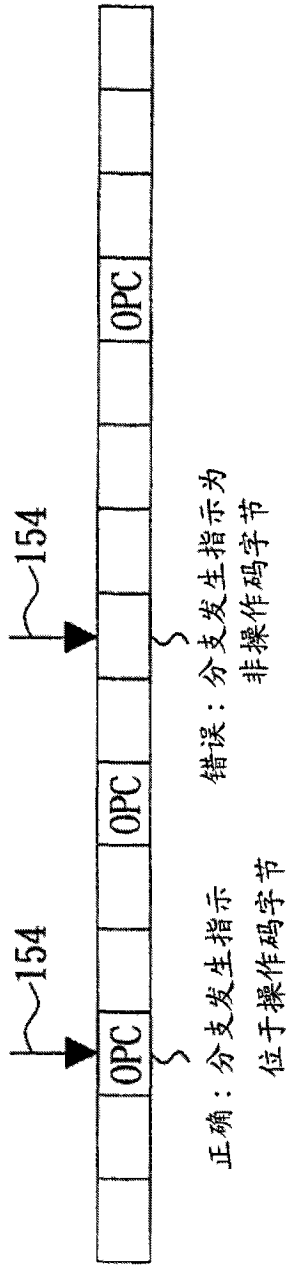

154
OPC
OPC
OPC
154
正确：分支发生指示
位于操作码字节
错误：分支发生指示为
非操作码字节

图 16

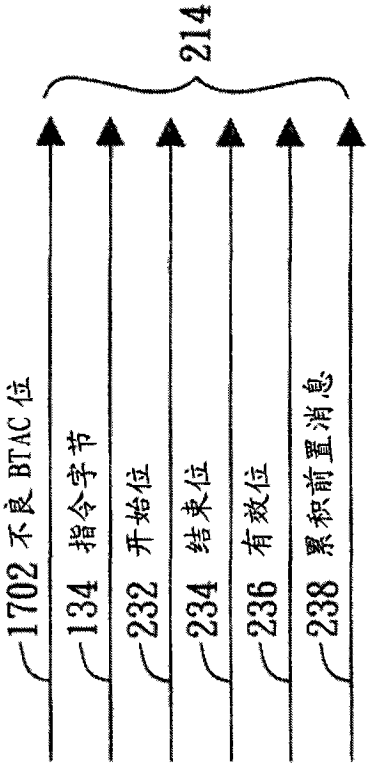

214
1702 不良 BTAC 位
134 指令字节
232 开始位
234 结束位
236 有效位
238 累积前置消息

图 17

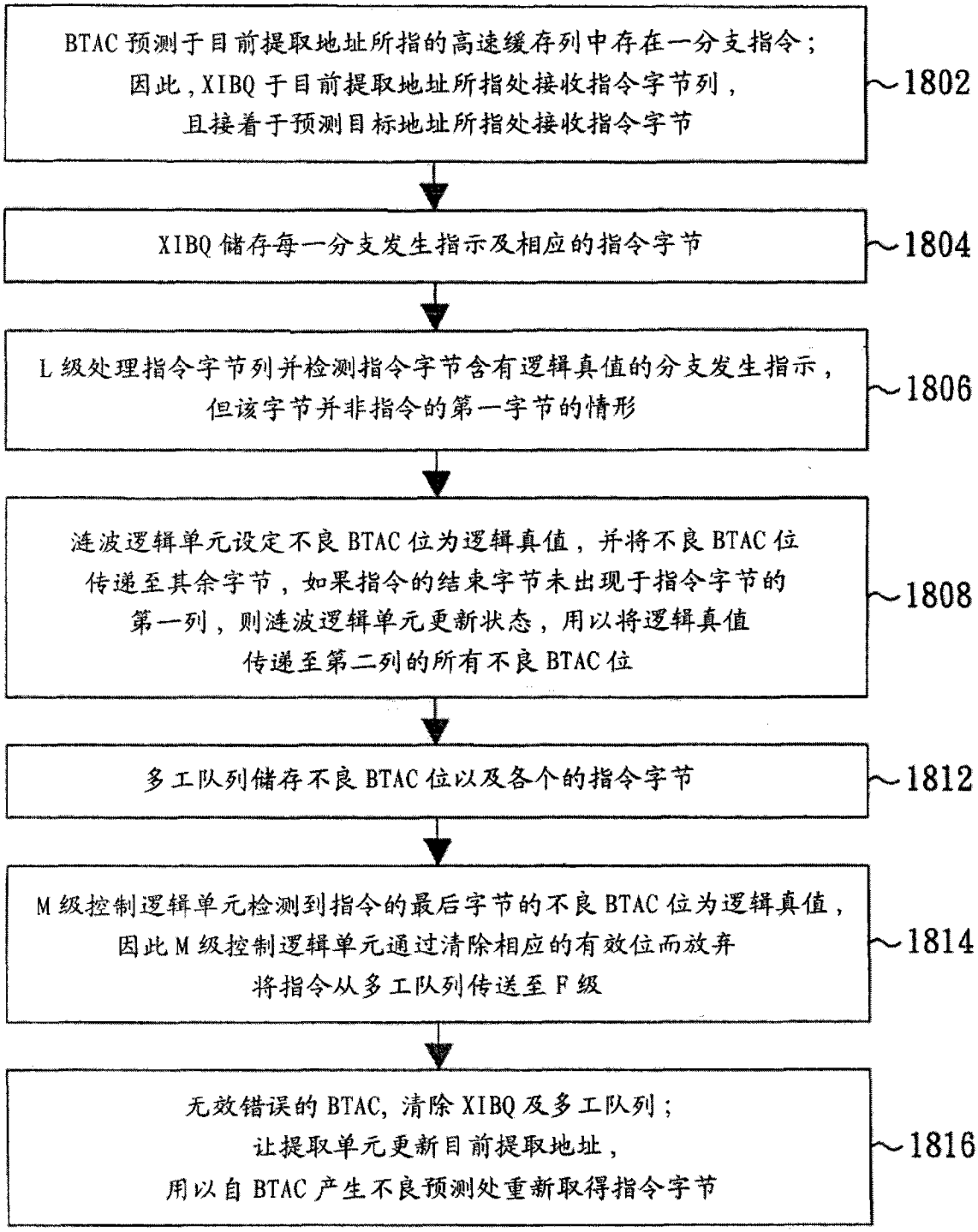

BTAC 预测于目前提取地址所指的高速缓存列中存在一分支指令；
因此，XIBQ 于目前提取地址所指处接收指令字节列，
且接着于预测目标地址所指处接收指令字节
1802
XIBQ 储存每一分支发生指示及相应的指令字节
1804
L 级处理指令字节列并检测指令字节含有逻辑真值的分支发生指示，
但该字节并非指令的第一字节的情形
1806
涟波逻辑单元设定不良 BTAC 位为逻辑真值，并将不良 BTAC 位
传递至其余字节，如果指令的结束字节未出现于指令字节的
第一列，则涟波逻辑单元更新状态，用以将逻辑真值
传递至第二列的所有不良 BTAC 位
1808
多工队列储存不良 BTAC 位以及各个的指令字节
1812
M 级控制逻辑单元检测到指令的最后字节的不良 BTAC 位为逻辑真值，
因此 M 级控制逻辑单元通过清除相应的有效位而放弃
将指令从多工队列传送至 F 级
1814
无效错误的 BTAC，清除 XIBQ 及多工队列；
让提取单元更新目前提取地址，
用以自 BTAC 产生不良预测处重新取得指令字节
1816

图 18

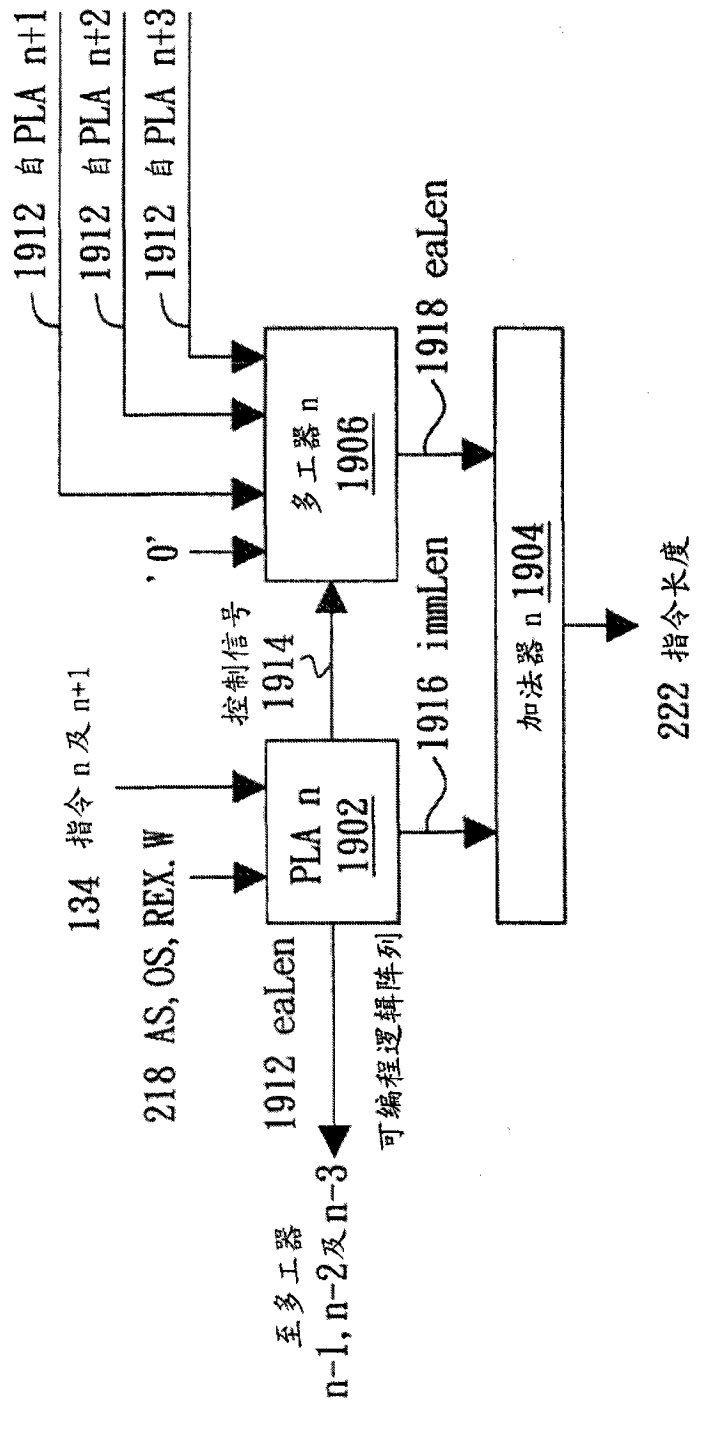

1912 自PLA n+1
1912 自PLA n+2
1912 自PLA n+3
'0'
多工器 n
1906
1918 eaLen
134 指令 n 及 n+1
控制信号
1914
218 AS, OS, REX. W
PLA n
1902
1916 immLen
加法器 n 1904
222 指令长度
1912 eaLen
至多工器
n-1, n-2及n-3
可编程逻辑阵列
选择'0': 当操作码，或 OF+ 操作码（不具 ModR/M)
选择 n+1: 当操作码 +ModR/M
选择 n+2: 当 OF+ 操作码 +ModR/M
选择 n+3: 当 OF+38/3A+ 操作码 +ModR/M

图 19

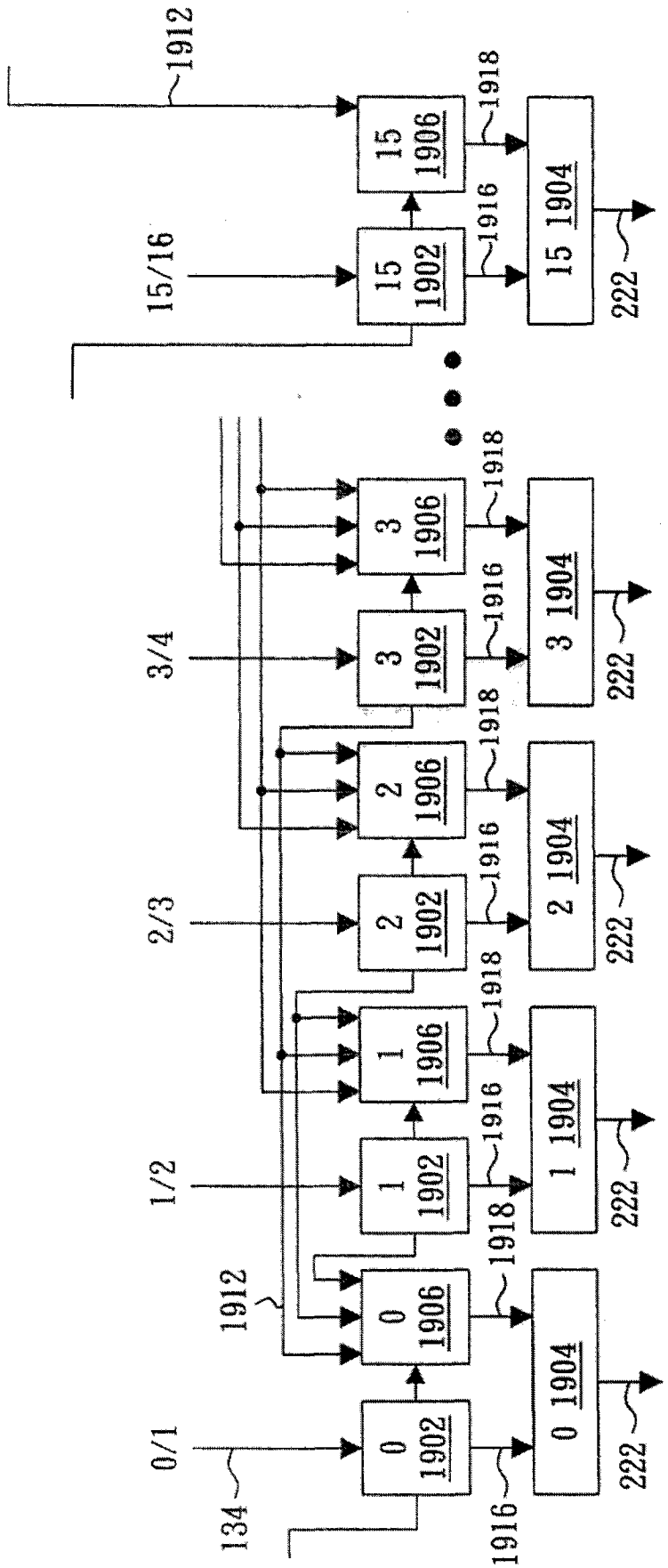

1912
15/16
15 1902
15 1906
1918
1916
15 1904
222
3/4
3 1902
3 1906
3 1904
2/3
2 1902
2 1906
2 1904
1/2
1 1902
1 1906
1 1904
0/1
0 1902
0 1906
0 1904
134

图 20

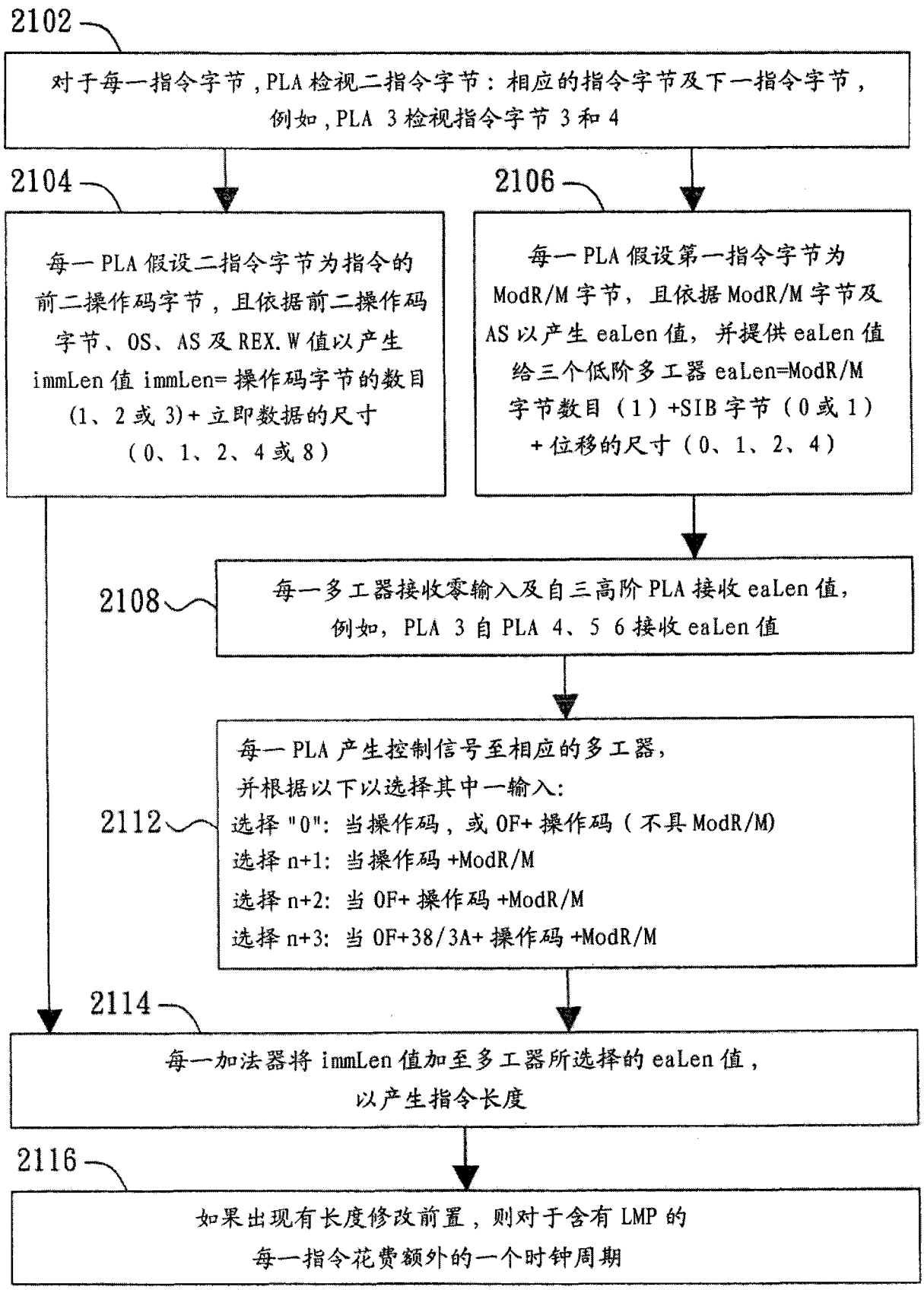

2102
对于每一指令字节，PLA 检视二指令字节：相应的指令字节及下一指令字节，例如，PLA 3 检视指令字节 3 和 4
2104
每一 PLA 假设二指令字节为指令的前二操作码字节，且依据前二操作码字节、OS、AS 及 REX.W 值以产生 immLen 值 immLen= 操作码字节的数目（1、2 或 3）+ 立即数据的尺寸（0、1、2、4 或 8）
2106
每一 PLA 假设第一指令字节为 ModR/M 字节，且依据 ModR/M 字节及 AS 以产生 eaLen 值，并提供 eaLen 值给三个低阶多工器 eaLen=ModR/M 字节数目（1）+SIB 字节（0 或 1）+ 位移的尺寸（0、1、2、4）
2108
每一多工器接收零输入及自三高阶 PLA 接收 eaLen 值，例如，PLA 3 自 PLA 4、5 6 接收 eaLen 值
2112
每一 PLA 产生控制信号至相应的多工器，并根据以下以选择其中一输入：
选择 "0"：当操作码，或 OF+ 操作码（不具 ModR/M）
选择 n+1：当操作码 +ModR/M
选择 n+2：当 OF+ 操作码 +ModR/M
选择 n+3：当 OF+38/3A+ 操作码 +ModR/M
2114
每一加法器将 immLen 值加至多工器所选择的 eaLen 值，以产生指令长度
2116
如果出现有长度修改前置，则对于含有 LMP 的每一指令花费额外的一个时钟周期

图 21